

Программирование
Excel с использованием VBA

2-Е ИЗДАНИЕ



Использование

макросов в Excel

O'REILLY®

 ПИТЕР®

С. Роман

Writing Excel Macros with VBA

Second Edition

Steven Roman

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Paris • Sebastopol • Taipei • Tokyo

Использование макросов в Excel

2-е издание

С. Роман



Москва • Санкт-Петербург • Нижний Новгород • Воронеж
Ростов-на-Дону • Екатеринбург • Самара • Новосибирск
Киев • Харьков • Минск

2004

ББК 32.973-018.2
УДК 681.3.06
Р69

Р69 **Использование макросов в Excel. 2-е изд. / С. Роман. — СПб.: Питер, 2004. — 507 с.: ил.**

ISBN 5-94723-584-6

Несмотря на мощные функциональные возможности, обеспечиваемые пользовательским интерфейсом Excel, существует ряд задач, выполнение которых возможно только программным путем. Книга «Использование макросов в Excel» представляет собой введение в программирование на Excel VBA, содержащее примеры решения различных практических задач, возникающих при работе в Excel.

Материал книги рассчитан на пользователей Excel, а также программистов, которые не знакомы с объектной моделью Excel. Наряду с разделами, посвященными разработке макросов для Excel 2002, здесь излагается вводный курс по написанию макросов и программ в среде Excel.

ББК 32.973-018.2
УДК 681.3.06

Права на издание получены по соглашению с O'Reilly.

Все права защищены. Никакая часть данной книги не может быть воспроизведена в какой бы то ни было форме без письменного разрешения владельцев авторских прав.

Информация, содержащаяся в данной книге, получена из источников, рассматриваемых издательством как надежные. Тем не менее, имея в виду возможные человеческие или технические ошибки, издательство не может гарантировать абсолютную точность и полноту приводимых сведений и не несет ответственности за возможные ошибки, связанные с использованием книги.

ISBN 0-596-00359-5 (англ.)
ISBN 5-94723-584-6

© 2002 O'Reilly&Associates, Inc.

© Перевод на русский язык, ЗАО Издательский дом «Питер», 2004

© Издание на русском языке, оформление, ЗАО Издательский дом «Питер», 2004

Краткое содержание

Предисловие	16
Глава 1. Введение	22
Часть I. Среда VBA	
Глава 2. Начальные сведения	28
Глава 3. Редактор Visual Basic: часть I	35
Глава 4. Редактор Visual Basic: часть II	46
Часть II. Язык программирования VBA	
Глава 5. Переменные, типы данных и константы	62
Глава 6. Функции и подпрограммы	81
Глава 7. Встроенные функции и операторы	89
Глава 8. Управляющие операторы	102
Часть III. Объектная модель Excel и приложения	
Глава 9. Объектные модели	110
Глава 10. Приложения Excel	121
Глава 11. События Excel	136
Глава 12. Пользовательские меню и панели инструментов	142
Глава 13. Встроенные диалоговые окна	162
Глава 14. Нестандартные диалоговые окна	167
Глава 15. Объектная модель Excel	181
Глава 16. Объект Application	188
Глава 17. Объект Workbook	216
Глава 18. Объект Worksheet	233
Глава 19. Объект Range	250
Глава 20. Сводные таблицы	311
Глава 21. Объект Chart	372
Глава 22. Смарт-теги	431
Приложения	
Приложение А. Объект Shape	436
Приложение Б. Получение сведений об установленных принтерах	448
Приложение В. Элементы управления командными панелями	451
Приложение Г. Значки для командных кнопок	461
Приложение Д. Разработка макросов Excel в среде других приложений	467
Приложение Е. Языки программирования высокого и низкого уровней	471
Приложение Ж. Новые объекты в Excel XP	481
Алфавитный указатель	495

Содержание

Предисловие 16

Предисловие ко второму изданию	17
Для кого предназначена эта книга	17
Структура книги	19
Текст книги и примеры кода	19
О примерах кода	20
Благодарности	21
От издательства	21
Об авторе	21

Глава 1. Введение 22

Выделение определенных ячеек	22
Настройка подписей на диаграммах	24
Изучение программирования в Excel	25

Часть I. Среда VBA

Глава 2. Начальные сведения 28

Что такое язык программирования?	28
Стиль программирования	29
Комментарии	30
Читаемость	30
Модульность	32

Глава 3. Редактор Visual Basic: часть I 35

Окно проекта	35
Имена проектов	37
Содержимое проекта	37
Окно Properties	39
Окно Code	40
Представления полного модуля и процедуры	40
Раскрывающиеся списки Object и Procedure	41
Объекты Workbook и Sheet	41
Стандартный модуль	41
Объект UserForm	42
Окно Immediate	43

Расположение окон	43
Пристыковка	45

Глава 4. Редактор Visual Basic: часть II 46

Управление интегрированной средой разработки	46
Общее управление	46
Справочная система	47
Создание процедур	48
Режимы выполнения, проектирования и прерывания	48
Ошибки	49
Ошибки, возникающие во время проектирования	50
Ошибки, возникающие во время компиляции	51
Ошибки, возникающие во время запуска программы	51
Логические ошибки	52
Отладка	53
Пошаговое выполнение программы	54
Просмотр выражений	56
Макросы	58
Запись макросов	58
Запуск макросов	59

Часть II. Язык программирования VBA

Глава 5. Переменные, типы данных и константы 62

Комментарии	62
Продолжение строки	62
Константы	63
Перечисления	63
Переменные и типы данных	65
Описание переменных	66
Зачем нужно подробное описание переменных	67
Численные типы данных	68
Логические типы данных	68
Строковые типы данных	69
Дата и время	69
Тип данных Variant	70
Объектные типы данных	70
Массивы	71
Стандарты именования	73
Область видимости переменных	74
Время жизни переменной	76
Инициализация переменных	79
Операторы VBA	79

Глава 6. Функции и подпрограммы 81

Вызов функций	81
Вызов подпрограмм	82
Параметры и аргументы	83
Необязательные аргументы	83
Именованные аргументы	84
Передача аргументов	85
Выход из процедуры	87
Область видимости процедур	87
Ссылки на проект	88
Уточненное имя процедуры	88

Глава 7. Встроенные функции и операторы 89

Функция MsgBox	90
Функция InputBox	92
Функции для обработки строк	92
Другие функции и операторы	95
Функции Is	95
Функция Immediate If	96
Функция Switch	96
Преобразования единиц измерения	97
Оператор Beer	98
Обработка ошибок	98
Оператор On Error	98
Объект Error	99
Оператор On Error GoTo 0	100
Оператор On Error Resume Next	100
Оператор Resume	100

Глава 8. Управляющие операторы 102

Оператор If...Then	102
Цикл For...Next	103
Выход из цикла For	103
Цикл For Each	104
Цикл Do	105
Оператор Select Case	106
Заключение	106
Функции, связанные с файлами	107
Функции, связанные со временем и датой	107
Функции форматирования	108

Часть III. Объектная модель Excel и приложения

Глава 9. Объектные модели 110

Объекты, свойства и методы	110
Свойства	110
Методы	111
Семейства объектов	111
Нумерация членов семейства	112
Иерархия объектной модели	113
Синтаксис объектной модели	114
Объектные переменные	115
Оператор With	116
Сокращение времени выполнения программы	116
Ссылка на объект	116
Ключевое слово Nothing	118
Оператор Is	118
Параметры объектной переменной по умолчанию	118
Глобальные параметры	119

Глава 10. Приложения Excel 121

Доступ к функциям приложений	122
Интерактивная работа с панелями инструментов и меню	123
Назначение макросов меню и панелям инструментов	125
Место хранения приложений	125
Загрузочная папка Excel	126

Шаблоны Excel	126
Настройки Excel	128
Примеры надстроек	133
Создание исходной книги	133
Настройка пользовательских меню	133
Выполнение функций надстройки	134
Заключительные шаги	135
Глава 11. События Excel	136
Свойство EnableEvents	136
События и объектная модель Excel	137
Доступ к процедуре обработки события	137
События объекта Worksheet	138
События объекта Workbook	138
События объекта Chart	139
События объекта Application	140
Глава 12. Пользовательские меню и панели инструментов	142
Меню и панели инструментов: общий обзор	142
Вид меню	142
Объект CommandBar	142
Элементы управления объекта CommandBar	143
Создание пункта меню	144
Семейство CommandBars	145
Создание новых меню и панелей инструментов	146
Элементы управления командной панели	147
Создание нового элемента управления командной панели	148
Встроенные элементы управления командной панели	149
Пример: создание меню	151
Пример: создание панели инструментов	152
Пример: новый элемент на существующем меню	154
Расширение приложения SRXUtils	154
Создание листа с данными	155
Настройка пользовательских меню	156
Дополнительные функции надстройки	158
Завершение работы с надстройкой	160
Глава 13. Встроенные диалоговые окна	162
Метод Show	165
Глава 14. Нестандартные диалоговые окна	167
Объект UserForm	167
Создание объекта UserForm	168
Элемент управления ActiveX	169
Код объекта UserForm	169
Стандартные элементы управления Excel	171
Пример: служебная программа ActivateSheet	172
Изменение процедуры ActivateSheet	173
Создание объекта UserForm	173
Создание кода для объекта UserForm	174
Проверка работы служебной программы	177
Элементы управления ActiveX на листе	177
Ссылка на расположенный на листе элемент управления	178
Программный способ добавления элементов управления на лист	180

Глава 15. Объектная модель Excel 181

Обзор объектной модели Excel	181
Перечисления Excel	184
Браузер объектов	186

Глава 16. Объект Application 188

Свойства и методы объекта Application	190
Свойства и методы, возвращающие дочерние объекты	192
Свойства и методы, влияющие на отображение информации	194
Свойства и методы, влияющие на функции Excel	195
Свойства и методы, связанные с событиями	196
Свойства и методы, связанные с вычислениями	198
Свойства и методы, связанные с файлами	199
Свойства и методы, влияющие на текущее состояние Excel	202
Свойства и методы, приводящие к выполнению различных операций	204
Прочие свойства и методы	208
Потомки объекта Application	212
Объект Name и семейство Names	212
Семейство Windows и объекты Window	214
Объект WorksheetFunction	215

Глава 17. Объект Workbook 216

Семейство Workbooks	216
Метод Add	216
Метод Close	217
Свойство Count	217
Свойство Item	217
Метод Open	218
Метод OpenText	218
Объект Workbook	221
Метод Activate	224
Метод Close	224
Свойство DisplayDrawingObjects	224
Свойство FileFormat (только чтение, тип Long)	224
Свойства Name, FullName и Path	225
Свойство HasPassword (только чтение, тип Boolean)	226
Свойство PrecisionAsDisplayed (чтение/запись, тип Boolean)	226
Метод PrintOut	226
Метод PrintPreview	226
Метод Protect	226
Свойство ReadOnly (только чтение, тип Boolean)	227
Метод RefreshAll	227
Метод Save	227
Метод SaveAs	227
Метод SaveCopyAs	228
Свойство Saved (чтение/запись, тип Boolean)	228
Потомки объекта Workbook	228
Объект CustomView	228
Семейство Names	229
Семейство Sheets	229
Семейство Styles и объект Style	229
Пример: сортировка листов книги	230

Глава 18. Объект Worksheet 233

Свойства и методы объекта Worksheet	233
Потомки объекта Worksheet	240

Функции защиты в Excel XP	243
Объект Protection	243
Объект AllowEditRanges	244
Объект UserAccess	245
Пример: вывод на печать листов	245
Создание объекта UserForm	246
Создание кода для объекта UserForm	247

Глава 19. Объект Range..... 250

Объект Range как семейство	251
Определение объекта Range	252
Свойство Range	253
Свойство Cells	254
Свойства Column, Row и Rows	255
Свойство Offset	257
Дополнительные свойства и методы объекта Range	258
Метод Activate	258
Метод AddComment	258
Свойство Address (только чтение, тип String)	258
Метод AutoFill	259
Метод AutoFilter	260
Метод AutoFit	262
Метод AutoFormat	262
Метод BorderAround	263
Метод Calculate	264
Метод Clear	264
Методы ColumnDifferences и RowDifferences	265
Свойства ColumnWidth и RowHeight	265
Свойства Width, Height, Top и Left	266
Метод Consolidate	267
Методы Copy и Cut	268
Метод CopyFromRecordset	269
Метод CreateNames	269
Свойство CurrentRegion	270
Метод Delete	270
Свойства Dependents и DirectDependents	270
Свойства Precedents и DirectPrecedents	271
Свойство End	271
Свойства EntireColumn и EntireRow	271
Методы Fill	272
Метод Find	272
Методы FindNext и FindPrevious	273
Свойства Formula и FormulaR1C1	274
Свойство FormulaArray	275
Свойство FormulaHidden (чтение/запись, тип Boolean)	276
Свойство HasFormula (только чтение)	276
Свойство HorizontalAlignment	276
Свойство IndentLevel и метод InsertIndent	276
Метод Insert	277
Свойство Locked	277
Методы и свойства, связанные со слиянием	277
Свойства Next и Previous	278
Свойство NumberFormat	278
Метод Parse	278
Метод PasteSpecial	279
Метод PrintOut	279
Метод PrintPreview	280

Метод Replace	280
Метод Select	281
Свойство ShrinkToFit	281
Метод Sort	281
Метод SpecialCells	282
Метод TextToColumns	283
Свойство Value	285
Свойство WrapText	285
Потомки объекта Range	286
Семейство Areas	286
Семейство Borders	287
Объект Border	288
Объект Characters	290
Объект Comment	291
Объект Font	291
Семейство FormatConditions	292
Объект Interior	293
Объекты PivotField, PivotItem и PivotTable	294
Объект QueryTable	294
Объект Validation	295
Пример: определение используемого диапазона	298
Пример: выделение ячеек по заданному признаку	300
Конструирование служебной программы	300
Конструирование диалогового окна	301
Написание кода	303

Глава 20. Сводные таблицы 311

Сводные таблицы	311
Мастер сводных таблиц	314
Метод PivotTableWizard	316
Объект PivotTable	319
Именованные поля данных	323
Итоговая программа	323
Свойства и методы объекта PivotTable	324
Свойства, возвращающие семейство Fields	325
Свойства, влияющие на итоговое значение	327
Возвращение части сводной таблицы	327
Метод PivotSelect и свойство PivotSelection	331
Дополнительные свойства и методы объекта PivotTable	335
Потомки объекта PivotTable	338
Объект PivotField	338
Свойства, связанные с методом AutoShow	340
Сортировка полей сводной таблицы	341
Основополагающие свойства	342
Выделение диапазонов	343
Перемещение полей сводной таблицы	343
Свойства Name, Value и SourceName	344
Группировка	344
Вычисление поля данных	345
Свойство CurrentPage	351
Свойство DataType	351
Свойства HiddenItems и VisibleItems	351
Свойство MemoryUsed	352
Свойство ServerBased	352
Свойство ShowAllItems	352
Метод Subtotals	353

Объект PivotCache	355
Обновление кэша сводной таблицы	355
Свойство MemoryUsed	356
Свойство OptimizeCache	356
Свойство RecordCount	356
Свойство SourceData	356
Свойство Sql	356
Объект PivotItem	356
Свойство DataRange	357
Свойство LabelRange	358
Свойство IsCalculated	358
Свойства Name, Value и SourceName	358
Свойство Position	358
Свойство RecordCount	359
Свойство ShowDetail	359
Свойство Visible	361
Объекты PivotCell и PivotItemList	361
Свойства ColumnItems, RowItems и DataField	361
Свойство CustomSubtotalFunction	362
Свойство PivotCellType	363
Свойства PivotTable, PivotField и PivotItem	363
Вычисляемые элементы и поля	364
Пример: вывод на печать сводной таблицы	367
Создание объекта UserForm	368
Создание кода для объекта UserForm	369

Глава 21. Объект Chart 372

Объекты Chart и ChartObject	372
Создание диаграммы	374
Создание диаграмм на отдельном листе	374
Создание встроенных диаграмм	375
Пример создания диаграммы	376
Порядок расположения по оси Z объектов ChartObject	379
Типы диаграмм	379
Свойство ChartType	379
Метод ApplyCustomType	382
Потомки объекта Chart	384
Семейство Axes	384
Объект Axis	386
Свойство AxisGroup	387
Заголовки осей и их форматирование	387
Свойство CategoryNames	389
Свойства CategoryType и BaseUnit	390
Свойства Crosses и CrossesAt	390
Отображение значений	391
Свойства, связанные с линиями сетки, и объект Gridline	392
Свойства, связанные с положением и размером осей	392
Свойства, связанные с временными метками	393
Свойства, связанные с единицами измерения	394
Свойство ReversePlotOrder	395
Свойство Type	396
Объект ChartArea	397
Объект ChartGroup	397
Объекты UpBars и DownBars	399
Объект DropLines	400
Объект HiLoLines	400
Семейство SeriesCollection и объект Series	400
Объект SeriesLines	400

Объект ChartTitle	401
Объект DataTable	402
Объект Floor	403
Объект Legend	403
Объект LegendEntry	404
Объект LegendKey	404
Объект PageSetup	405
Объект PlotArea	405
Объект Series	406
Добавление новых рядов данных	407
Объект DataLabel	408
Объект Point	410
Свойства и методы объекта Chart	412
Метод ChartWizard	414
Метод Export	416
Метод PrintOut	416
Пример: прокрутка списка типов диаграмм	417
Пример: вывод на печать встроенной диаграммы	419
Создание объекта UserForm	420
Создание кода для объекта UserForm	421
Пример: установка меток рядов данных	424

Глава 22. Смарт-теги 431

Что такое смарт-теги?	431
Объект SmartTagRecognizer	432
Объект SmartTag	433
Объект SmartTagAction	433
Объект SmartTagOptions	434

Приложения

Приложение А. Объект Shape 436

Объект Shape	436
Z-порядок	436
Создание форм	438
Объект TextFrame	441
Объект FillFormat	441
Примеры	441
Объекты Diagram, DiagramNode и DiagramNodeChildren	445

Приложение Б. Получение сведений об установленных принтерах 448

Приложение В. Элементы управления командными панелями 451

Встроенные элементы управления командными панелями	451
--	-----

Приложение Г. Значки для командных кнопок 461

Приложение Д. Разработка макросов Excel в среде других приложений 467

Ссылки на объектную модель Excel	467
Ссылки на объект Application в Excel	467
Альтернативный подход	469

**Приложение Е. Языки программирования высокого
и низкого уровней 471**

BASIC	473
Visual Basic	474
Языки С и С++	475
Visual C++	476
Pascal	477
FORTRAN	478
COBOL	478
LISP	479

Приложение Ж. Новые объекты в Excel XP 481

Объект AllowEditRange	481
Объект AutoRecover	482
Объект CalculatedMember	482
Объект CellFormat.....	483
Объект CustomProperty	483
Объекты Diagram, DiagramNode и DiagramNodeChildren	484
Объект Error	485
Объект ErrorCheckingOptions	486
Объект Graphic	487
Объекты IRTDServer и IRTDUpdateEvent	488
Объекты PivotCell и PivotItemList	488
Объект Protection	489
Объект RTD	490
Объекты, связанные со смарт-тегами	490
Объект Speech	491
Объект SpellingOptions	492
Объект Tab	492
Объект UsedObjects	493
Объекты UserAccessList и UserAccess	493
Объект Watch	593

Алфавитный указатель 495

Предисловие

Данная книга предназначена для тех, кто хотел бы научиться программировать в Excel 8 (для Office 97) и Excel 9 (для Office 2000).

Начать изучение материала следует с ответа на вопрос: «А зачем кому-то вообще программировать в Microsoft Excel?» Разумеется, чтобы получить дополнительные возможности при работе с этим внушительным приложением. Как вы вскоре убедитесь, многие вещи вы просто не можете делать на уровне пользовательского интерфейса, то есть с помощью меню и диалоговых окон. Конкретные примеры вы найдете уже в первой главе.

Эта книга представляет собой инструкцию для программирования в Excel с помощью пакета VBA (Visual Basic for Applications — Visual Basic для прикладных программ). Однако следует помнить, что она не является энциклопедией по программированию в Excel, а всего лишь дает основные сведения по данной тематике, которых вполне достаточно для последующего самостоятельного изучения. После прочтения данной книги вам не потребуется других пособий, кроме окна просмотра и справочного файла Excel VBA или хорошего справочника по этой теме.

Мой опыт показывает, что книги, посвященные введению в программирование (как, впрочем, и большинство книг по компьютерной тематике), требуют большого объема самостоятельной работы. Понятия раскрываются медленно, сопровождаемые громоздкими примерами и не относящимися к делу анекдотами, которые находит забавными только автор издания. Все это затрудняет восприятие фактов. Откровенно говоря, меня подобный непрофессионализм приводит в бешенство. Я считаю, что просто недопустимо на 400 страниц информации давать еще 600 страниц рассуждений ни о чем.

Без сомнения, для написания книг требуются более профессиональные авторы, однако найти писателя, который не только обладает знаниями в определенной области, но и имеет педагогическую манеру их изложения, достаточно сложно. Здесь я должен спешно добавить, что существуют замечательные авторы, пишущие на данную тему, но их явно недостаточно. Кроме того, издатели любят вдохновлять на создание томов в 1000 и более страниц, потому что считают, что только толстая книга может занять достойное место на полке. Я содрогаюсь при мысли,

что это и в самом деле может быть так. Но, к счастью, в издательстве O'Reilly подобный подход не практикуется.

В книге «Использование макросов в Excel» вы не найдете посторонних рассуждений. Впрочем, то же самое можно сказать о любой из моих книг. Достаточно быстро произойдет переход от вводных понятий к программированию на языке Visual Basic и к рассмотрению объектного моделирования в Excel. Это поистине безграничная тема, поэтому написать книгу, которая исчерпывающе разъясняла бы все тонкости, просто невозможно. Тем не менее существенные моменты языка VBA и работы в Excel описаны четко и понятно, поэтому в процессе чтения вы получите знания об объектном моделировании в Excel, достаточные для написания эффективных работающих программ.

Я попытался использовать мой преподавательский опыт, насчитывающий около 20 лет, и опыт, приобретенный за 30 лет писательской деятельности, чтобы создать обучающее пособие. Надеюсь, эта книга будет прочитана, и даже не один раз, и вы найдете тут много полезных ссылок.

Предисловие ко второму изданию

После появления версии Excel 10 (также называемой Excel XP) потребовалось обновление книги. В основном это эволюционный шаг вперед по сравнению с предыдущей версией, но некоторые интересные новые функции требуют особого внимания, например поддержка смарт-тегов.

Объектная модель Excel 10 имеет 37 новых объектов, включая 266 новых членов. Появились также 180 новых членов ранее существовавших объектов. В этой книге рассказывается о большей части основных объектов. На рис. 0.1 показана иерархия объектов в Excel XP. Это изображение получено в программе Object Model Browser. Дополнительную информацию об этой программе можно получить на моем сайте <http://www.romanpress.com>.

Для кого предназначена эта книга

В качестве вводного пособия в программирование в Excel эта книга предназначена двум основным группам читателей.

- Пользователи Excel, которые не занимались программированием, но хотели бы его освоить. В эту категорию попадают пользователи, которые оценили возможности приложения Excel и хотели бы использовать преимущества дополнительных функций или просто узнать более простой путь выполнения многих задач.
- Пользователи, знакомые с программированием (например, на языке Visual Basic, VBA, BASIC, C, C++ и т. п.), но не имеющие представления об объектном моделировании в Excel. В этом случае данную книгу можно использовать для получения дополнительных знаний по языку VBA и изучения объектной модели в Excel.

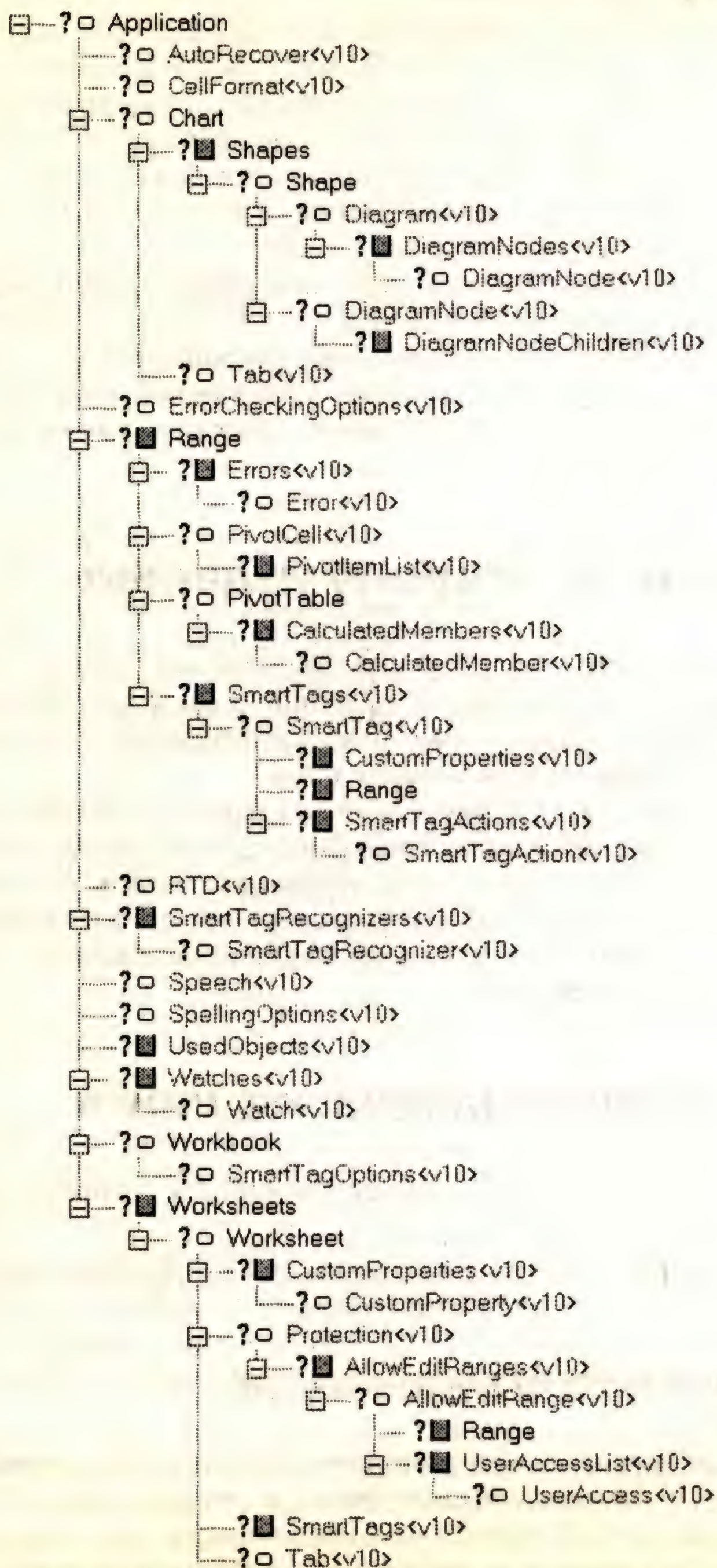


Рис. 0.1. Новые объекты в иерархии объектов Excel XP

Структура книги

Книга «Использование макросов в Excel» состоит из 22 глав, которые поделены на четыре части.

Вводная глава 1 доказывает необходимость изучения программирования на примерах задач, которые могут быть решены с его помощью. Глава 2 является введением в программирование и в язык VBA. В главах 3 и 4 рассказывается об интегрированной среде разработки (IDE — Integrated Development Environment) языка VBA, в которой вам предстоит работать.

Вторая часть книги посвящена введению в язык VBA и компонентам этого языка, общим с Microsoft Visual Basic и рядом основных приложений Microsoft, включая Word, Excel, PowerPoint, Access и продукты других производителей. Отдельные главы посвящены переменным, типам данных и константам (глава 5), функциям и процедурам (глава 6), встроенным функциям и операторам (глава 7) и управляющим операторам (глава 8).

В третьей части книги рассказывается об объектной модели Excel и о создании различных приложений. Рассмотрение начинается в главе 9, затем объясняется, из чего состоят приложения Excel (глава 10), что собой представляют события Excel (глава 11), как создать меню и панели инструментов в Excel (глава 12) и как работать со встроенными и нестандартными диалоговыми окнами (главы 13 и 14). Такой порядок следования тем выбран потому, что рассказ о создании приложений невозможен без информации обо всех элементах, формирующих эти приложения.

Последние главы посвящены исключительно объектной модели Excel, которая определяет, доступ к каким элементам данного приложения (книги, листы, диаграммы, ячейки и т. п.) можно получить программным путем. В главе 15 дается обзор объектной модели, а в последующих главах детально рассказывается об ее основных объектах. Так, глава 16 посвящена объекту Application, который представляет собственно приложение Excel, глава 17 — объекту Workbook, демонстрирующему книгу Excel, глава 18 — объекту Worksheet, представляющему листы Excel, глава 19 — объекту Range, который представляет наборы ячеек в книге, глава 20 — объекту PivotTable, а глава 21 — объекту Chart. В последней главе книги рассказывается о смарт-тегах. В конце большинства глав я постарался привести нужные примеры кода.

Приложения содержат многообразный набор сопутствующих материалов, включая обсуждение объекта Shape, который позволяет добавлять иллюстрации на лист Excel, определение принтеров, доступных для вашей системы (это не так просто, как может показаться на первый взгляд), и способов программирования Excel из других приложений (например, Word, Access или PowerPoint). Существует также приложение, содержащее краткий обзор языков программирования.

Текст книги и примеры кода

В процессе чтения данной книги вы найдете множество коротких примеров кода, иллюстрирующих различные понятия. Мне кажется, что понять смысл происходящего на небольшой части программы намного проще, чем путем изучения длинного запутанного кода. Работа с длинными программами осложняется тем, что изменение пары строк может оказать влияние на остальные части кода вплоть до

того, что программа перестанет работать. А вам в итоге придется тратить время на поиск причины сбоя.

Я рекомендую вам испытывать на практике все примеры кода и экспериментировать с ними. Это наилучший способ изучения нового материала. Но для экспериментов желательно использовать отдельную книгу, которую всегда можно удалить.

Напоследок стоит сделать еще одно замечание по поводу примеров кода. В конце концов, на их основе вам предстоит учиться написанию программ на языке VBA. Но в процессе этого обучения часто будет возникать ситуация, когда, образно выражаясь, «телега находится впереди лошади». Ведь практически невозможно дать пример одного объекта с его свойствами и методами без ссылки на другие объекты, которые будут обсуждаться в последующих главах книги. Честно говоря, я не вижу способа полностью обойти эту проблему. Впрочем, вы всегда можете воспользоваться ссылками на последующие главы.

О примерах кода

Все примеры кода, приведенные в данной книге, были проверены лично мной, редактором Роном Петруша и техническим консультантом Мэттом Чайлдсом. Кроме того, код был проверен на нескольких компьютерах с различными операционными системами и по меньшей мере два раза — во время написания книги и во время подготовки к публикации.

К сожалению, все мы сталкивались с отклонением поведения некоторых вариантов кода от ожидаемого (код выполнялся некорректно или не работал вообще) и с некоторой непоследовательностью (код по-разному срабатывал в различных операционных системах). Иногда случалось и такое, что при тестировании одного и того же куска кода с одними и теми же параметрами разные люди получали разные результаты. Более того, я иногда был не в состоянии продублировать свои собственные результаты через некоторый промежуток времени.

Думаю, что с учетом сложности приложения Excel и человеческой склонности к ошибкам такие вещи не должны вызывать удивления. Я могу с ходу назвать пару причин возникновения подобных ситуаций:

- Иногда для корректной работы кода необходимо соблюдать дополнительные условия, которые не указаны в документации. Например, нигде не сказано — по крайней мере, я не нашел упоминаний об этом, — что для размещения оси на диаграмме до указания расположения данных для этой оси нельзя использовать метод `HasAxis`. Мне такой подход напоминает поговорку о телеге, которая находится впереди лошади, но в данном случае вопрос не в этом. Попытка действовать подобным образом приведет к появлению сообщения об ошибке: «Метод `HasAxis` объекта `Chart` не работает». Эта информация, по крайней мере, поможет понять, в чем суть проблемы. Вам не придется гадать, почему код не работает.
- Компьютеры эволюционируют с течением времени. При установке любого приложения, пусть даже не связанного с Excel, существует вероятность, что файл DLL или любой другой системный файл будет заменен более новым. К сожалению, появление новой версии еще не означает лучшей работе. Это может стать причиной некорректной работы примеров кода, хотя и не служит оправданием такого поведения системы.

Все это написано для того, чтобы предупредить вас о возможных проблемах при экспериментах с кодом. Я попытался упомянуть о проблемах, с которыми сталкивался сам, но вы можете обнаружить другие недостатки. Впрочем, в большинстве случаев причиной ошибки является невнимательность пользователя и опечатки при наборе кода.

Я надеюсь, эта книга будет вам полезна. Вы можете посетить мой сайт <http://www.romanpress.com>. Примеры кода, приведенные в данной книге, можно также найти на сайте издательства по адресу <http://www.oreilly.com/catalog/exlmacro2>.

Благодарности

Я хотел бы выразить самые искренние благодарности Рону Петруша, моему редактору издательства O'Reilly. Как и в случае с другими моими книгами, Рон оказал существенную помощь. Он является одним из самых лучших среди тех редакторов, с которыми мне приходилось работать в течение 17 лет писательской деятельности.

Также я признателен Мэтту Чайлдсу, который на высоком уровне выполнил просмотр текста книги.

От издательства

Ваши замечания, предложения и вопросы отправляйте по адресу электронной почты comp@piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

Подробную информацию о наших книгах вы найдете на веб-сайте издательства <http://www.piter.com>.

Об авторе

Стивен Роман (Steven Roman) — заслуженный профессор математики из Университета штата Калифорния в Фуллертоне. Он преподавал и в ряде других университетов, в том числе и в Массачусетском технологическом институте, в Калифорнийском университете в Санта-Барбаре и в Университете Южной Флориды. Степень бакалавра доктор Роман получил в Калифорнийском университете в Лос-Анджелесе, а степень доктора — в Университете города Вашингтон. Он является автором свыше 35 книг по математике и вычислениям на персональной ЭВМ. Он также написал ряд программных приложений, в том числе и браузер объектов (Object Model Browser). Более подробную информацию о его книгах, статьях и программных продуктах можно получить на сайте <http://www.romanpress.com>.

1

Введение

Приложение Microsoft Excel обладает необычайной гибкостью и широчайшими возможностями. Однако, несмотря на мощный инструментарий, все же существуют задачи, которые невозможно выполнить с помощью пользовательского интерфейса Excel или же это сопряжено с определенными сложностями. В данном случае на помощь приходит программирование.

Рассмотрим два примера из моей практики.

Выделение определенных ячеек

В интерфейсе Excel невозможно выделение ячеек по определенному критерию. Например, нельзя выделить ячейки, значения которых принадлежат диапазону от 0 до 100 или ячейки с датой после 1 января 1998 года. Невозможно выделить в столбце ячейки, значение которых не совпадает со значением их непосредственных предшественников. Такая возможность была бы очень полезна в случае необходимости выделения уникальных значений в отсортированном столбце, как показано на рис. 1.1.

12	Denver
13	Denver
14	Denver
15	Denver
16	Los Angeles
17	Los Angeles
18	New York
19	New York
20	New York
21	New York
22	New York
23	Portland
24	Portland
25	Seattle
26	Seattle

Рис. 1.1. Выделение уникальных значений

Меня много раз спрашивали о существовании подобных способов выделения ячеек. По этой причине я решил написать служебную программу. Соответствующее диалоговое окно показано на рис. 1.2. Пользователю достаточно указать тип сравниваемых значений (например, число, дата или текст) и критерии выбора. При желании можно задать несколько критериев. Время показало, насколько полезна такая программа.

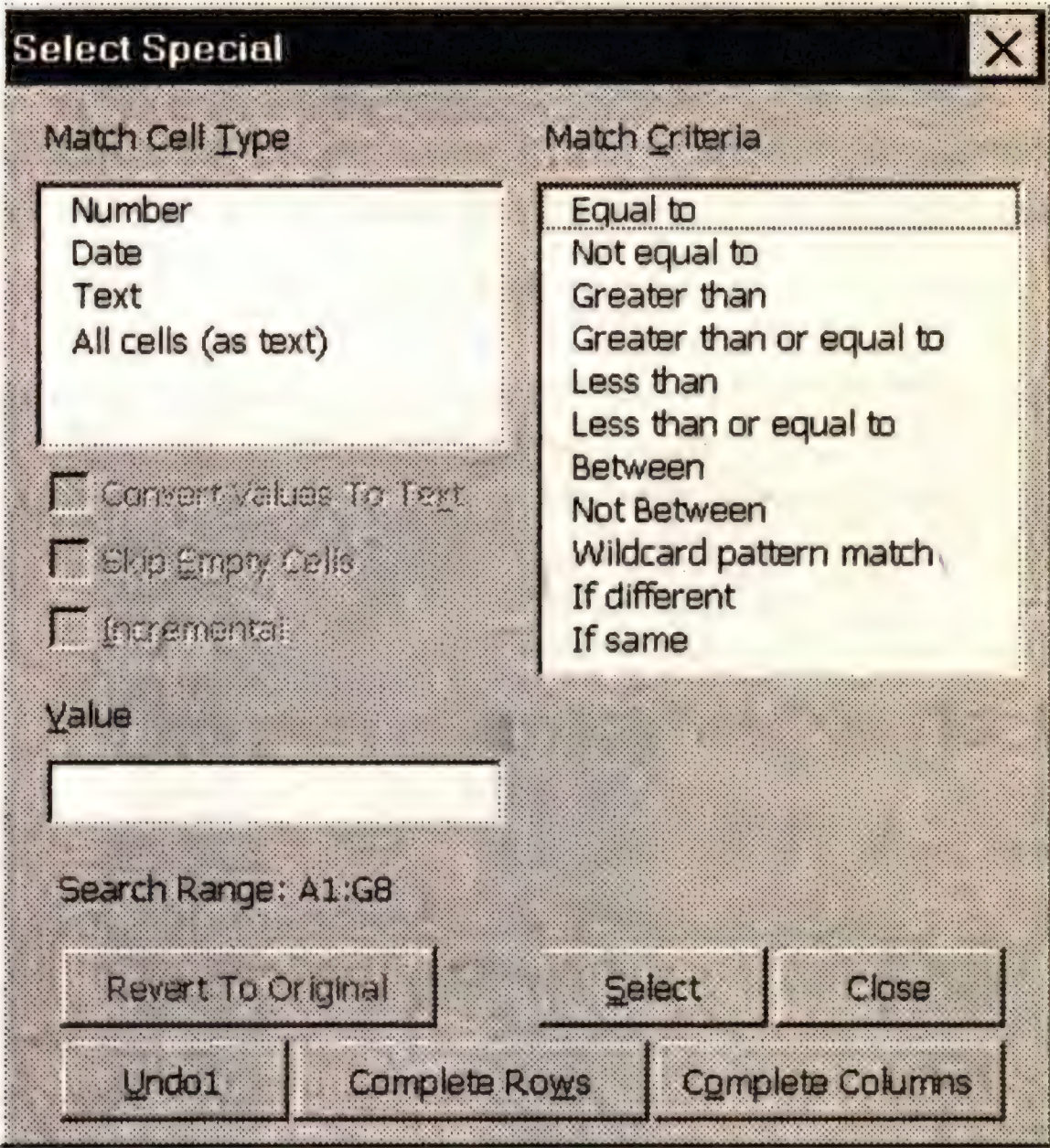


Рис. 1.2. Диалоговое окно служебной программы Select Special

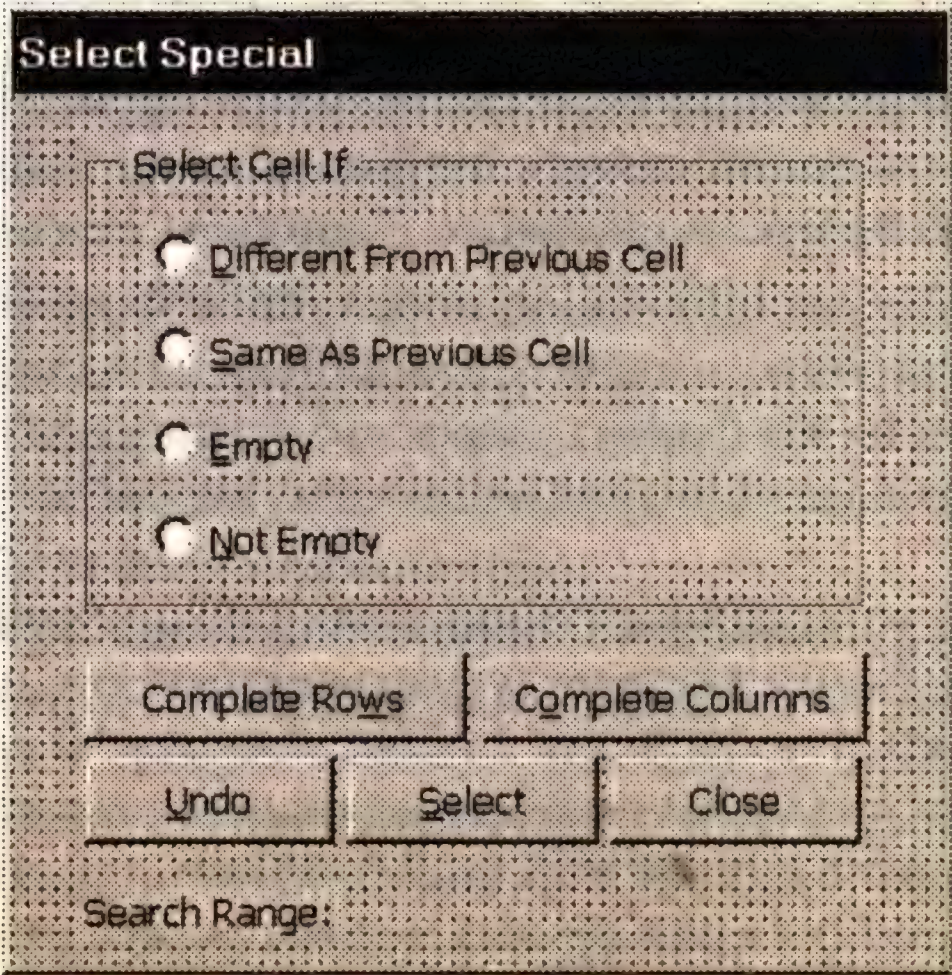


Рис. 1.3. Диалоговое окно Выделение по признаку

В процессе чтения данной книги вам предстоит разработать более простую версию программы, диалоговое окно которой показано на рис. 1.3. Также с помощью полученных значений вы сможете превратить его в диалоговое окно, изображенное на рис. 1.2.

Настройка подписей на диаграммах

Наверное, вы уже знаете, что подписи на диаграмме можно редактировать по отдельности. Для перехода в режим редактирования подписи, показанный на рис. 1.4, достаточно дважды щелкнуть на ней, выдерживая небольшую паузу между щелчками.

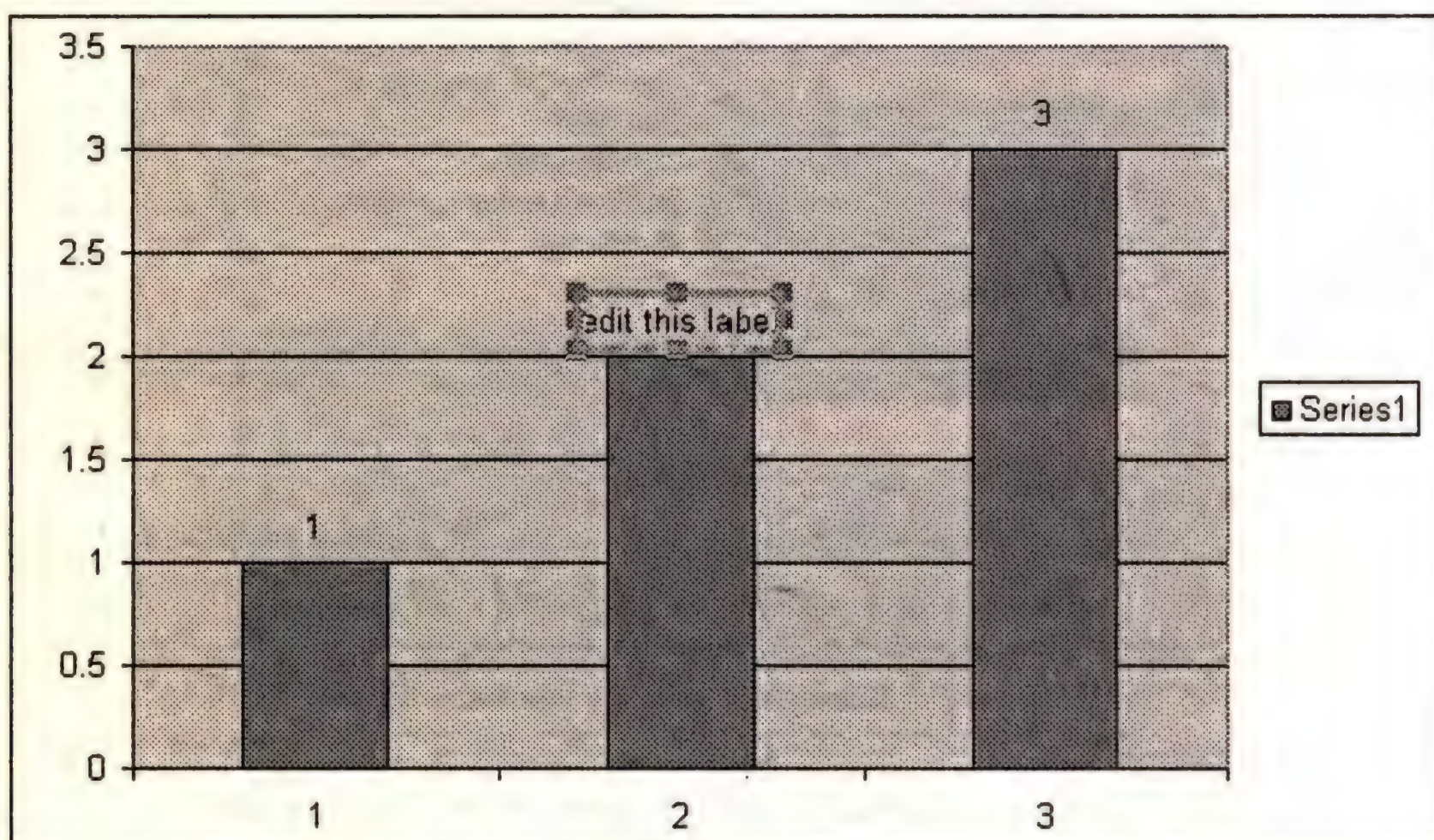


Рис. 1.4. Подписи данных в режиме редактирования

В этом режиме можно менять текст подписи (что приводит к разрыву всех связей) или устанавливать новые связи с ячейками листа. Данная задача легко выполняется и с помощью программирования. Например, строка `ActiveChart.SeriesCollection(1).DataLabels(2).Text = "MyChartSheet!R12C2"` связывает подпись для второго элемента ряда данных со значением в ячейке B12. Обратите внимание на то, что формула должна быть записана в нотации R1C1. (Сейчас не стоит акцентировать внимание на деталях, так как они будут обсуждаться в главе 21.)

К сожалению, для установления связи между подписями ряда данных и соответствующими им ячейками листа не существует простого способа. Вам придется вручную устанавливать эту связь для каждой подписи. В главе 21 будет создана служебная программа, диалоговое окно которой показано на рис. 1.5.

В этом окне находится список всех рядов данных для выбранной диаграммы. Пользователь может выбрать ряд данных и затем определить как диапазон ячеек, в дальнейшем связываемых с подписями, так и диапазон ячеек, значения которых

будут в эти подписи просто скопированы. В последнем случае связь между значениями ячеек и подписями не устанавливается и, соответственно, изменения диапазона значений этих ячеек не отображаются на диаграмме. В зависимости от вашего желания форматирование может быть связано с форматированием ячейки или просто скопировано.

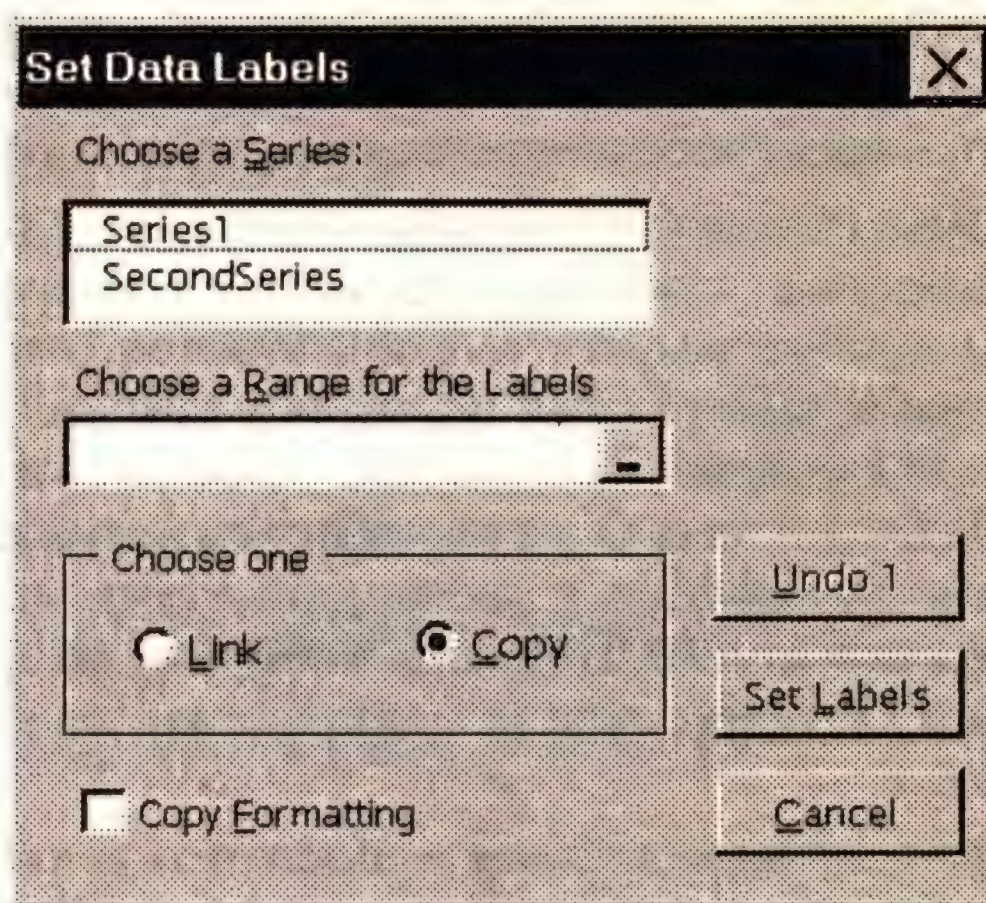


Рис. 1.5. Диалоговое окно Set Data Label

Я надеюсь, что эти примеры показали вам, насколько полезным может быть программирование в Excel. С его помощью выполняются банальные задачи, например автоматическая распечатка диаграмм, сортировка ячеек в алфавитном порядке и т. п.

Изучение программирования в Excel

В общем случае обучение программированию в Excel можно разбить на следующие стадии.

○ Редактор Visual Basic

Для начала вам требуется получить сведения о среде, в которой осуществляется программирование в Excel. Это так называемый редактор Visual Basic, или интегрированная среда разработки Excel VBA (IDE — Integrated Development Environment). Вы познакомитесь с ним в главах 3 и 4.

○ Основы программирования на языке VBA

Затем вам потребуется получить основные сведения о языке программирования, который используется в Excel. Он называется Visual Basic для прикладных программ (VBA — Visual Basic for Applications). Разумеется, этот язык используется и в прочих приложениях пакета Microsoft Office XP: Access, Word, PowerPoint. Приложения, использующие VBA таким способом, носят название *хост-приложений* (*host applications*) для VBA. Существует также ряд продуктов, не имеющих отношения к Microsoft, в основе которых лежит язык

программирования VBA. Наиболее примечательным из них является Visio, приложение для работы с векторной графикой. Кроме того, VBA используется в автономной программной среде, называемой Visual Basic (VB).

Обсуждение основ языка VBA вы найдете в главах 5–8.

○ Объекты и объектная модель Excel

Все хост-приложения VBA (Word, Access, Excel, PowerPoint, Visual Basic) дополняют основы этого языка объектными моделями, предназначенными для обработки объектов, присущих конкретному приложению.

Например, Excel VBA включает объектную модель Excel, предназначенную для работы с книгами, листами, ячейками, строками, столбцами, диапазонами, диаграммами, сводными таблицами и т. п., а объектная модель Word работает с документами, шаблонами, параграфами, шрифтами, заголовками, таблицами и т. п. В Access VBA входят две объектные модели — объектная модель Access и объектная модель DAO (Data Access Objects — объекты доступа к данным). Они позволяют программисту работать с такими объектами, как базы данных, таблицы, запросы, формы и отчеты. Получить информацию об объектных моделях Word, Access и DAO можно в других моих книгах, также выпущенных издательством O'Reilly.

Словом, для программирования в Excel пользователь должен иметь общее представление об объектных моделях и быть знаком с объектной моделью Excel в частности. Обсуждение этой темы начнется в главе 9 и продолжится в последующих главах книги.

Объектная модель Excel по сложности и размеру близка к объектной модели Word и включает почти 200 различных объектов.

Чтобы сложность объектной модели Excel не слишком вас обескураживала, следует отметить, что изучать мы будем только небольшое число объектов, без которых невозможно программирование в Excel VBA. Скоро вы убедитесь в том, что основные действия выполняются над семью объектами: Application, Range, WorksheetFunction, Workbook, Worksheet, PivotTable и Chart.

Чтобы дать вам возможность получить общее представление об объектной модели Excel и познакомиться с деталями ее использования, я написал специальное приложение для просмотра объектов (object browser). Более детальную информацию можно получить на сайте <http://www.romanpress.com>.

Даже если вы собираетесь работать в Excel ради получения прибыли, я надеюсь, что вы получите удовольствие, увидев, насколько расширяются возможности данного приложения после перехода на уровень программирования. А благодаря тому, что для доступа к объектной модели Excel используется язык VBA, после чтения данной книги вы будете иметь некоторое представление о программировании в Word, Access и PowerPoint.

Часть I. Среда VBA

- Глава 2. Начальные сведения
- Глава 3. Редактор Visual Basic: часть I
- Глава 4. Редактор Visual Basic: часть II

2

Начальные сведения

Начнем с общих сведений о программировании и языках, чтобы дать возможность читателю понять, о чем эта книга. Язык VBA является всего лишь одним из языков программирования, и любой желающий им воспользоваться должен четко представлять себе область его применения. Однако мы не будем задерживаться на второстепенных вопросах. Цель этой главы — дать обзор процесса написания программ языков программирования, который был бы интересен не только читателям, не имеющим опыта в данной области, но и тем, кто знает, о чем идет речь.

Что такое язык программирования?

Попросту говоря, язык программирования является специальным крайне ограниченным языком, который на определенном уровне понимается компьютером. В зависимости от назначения существующих языков их можно приблизительно разделить на три группы:

- Языки, управляющие компьютером на низком уровне, то есть воздействующие на операционную систему или даже на аппаратное обеспечение. Это так называемые *языки низкого уровня*, ярким представителем которых является Assembler.
- Языки, предназначенные для создания индивидуальных приложений, например Microsoft Excel. К *языкам высокого уровня* относятся BASIC, COBOL, FORTRAN, Pascal, C, C++ и Visual Basic.
- Языки, управляющие прикладными программами, например Microsoft Excel. К этим, так называемым *языкам прикладного уровня* относятся Excel VBA, Word VBA и PowerPoint VBA.

Данные термины достаточно условны и могут использоваться в других ситуациях. Однако никто не будет оспаривать тот факт, что некоторые языки используются на более низком уровне по сравнению с остальными.

Мир компьютеров заполнен сотнями языков программирования. Часть из них была разработана для определенных компьютеров, часть — для определенных типов приложений. В табл. 2.1 собраны названия языков программирования и их общего назначения.

Языки программирования имеют различный синтаксис. Некоторые из них читаются проще, чем остальные. В качестве примера в табл. 2.2 показан процесс при-

сваивания значения 5 переменной X в различных языках. Обратите внимание на то, насколько различается синтаксис даже в такой простой задаче.

Таблица 2.1. Языки программирования

Язык	Назначение
ALGOL	Результат попытки разработать универсальный язык
BASIC	Простой, легко изучаемый язык, созданный для начинающих
C, C++	Очень мощные языки, позволяющие полноправно управлять компьютером
COBOL	Язык для бизнес-приложений
FORTRAN	Язык для научного программирования и решения числовых задач большого объема
Lisp	Язык для обработки списков (используется искусственным интеллектом)
Pascal	Язык, на примере которого студентов обучают «правильному» программированию
SIMULA	Язык для моделирования физических явлений
Smalltalk	Язык объектно-ориентированного программирования
Visual Basic	Версия языка BASIC, предназначенная для создания приложений Windows
Visual C++	Версия языка C++, предназначенная для создания приложений Windows

Таблица 2.2. Присваивание значения в различных языках

Язык	Выражение присваивания
APL	X <- 5
BASIC	LET X = 5 OR X = 5
BETA	5 -> X
C, C++	X = 5
COBOL	MOVE 5 TO X
FORTRAN	X = 5
J	X =. 5
LISP	(SETQ X 5)
Pascal	X := 5
Visual Basic	X = 5

Для сравнения Visual Basic с другими основными языками программирования обратитесь к Приложению Е данной книги. Оно содержит краткое описание ряда языков с примерами программ.

Стиль программирования

Понятие хорошего стиля программирования, конечно же, субъективно. Вероятно, проще всего научиться хорошо программировать, изучая примеры, о которых желательно помнить в процессе написания программ.

В рамках данной книги, видимо, не стоит детально обсуждать стили программирования. Однако необходимо сформулировать два правила, которых следует придерживаться при написании программы:

- На первом месте должна быть читаемость кода.
- Программа должна быть снабжена многочисленными *значимыми* комментариями.

Комментарии

Начнем обсуждение со второго правила. Переоценить значение комментариев в программе невозможно, особенно если ее объем достаточно велик.

Проблема в том, что хорошие программы используются месяцами и даже годами. Неизбежно наступит момент, когда программист захочет внести изменения в собственный код, например добавить новую функцию или исправить ошибки. Однако в отличие от текста, написанного на обычном, разговорном языке, текст программы не так-то просто прочесть. Ничего удивительного нет в том, что программист не сможет понять (или даже не узнает) код, созданный несколько месяцев или лет назад. Именно поэтому программу желательно снабжать комментариями.

Я хотел бы подчеркнуть, что комментирование кода — это такое же искусство, как и его написание. Мне часто приходилось видеть подобные комментарии:

```
' Присвоим X значение 5  
X = 5
```

Этот комментарий совершенно бесполезен, потому что смысл кода и так ясен, и не стоило тратить время и место на его написание. Впрочем, в учебных пособиях (к которым относится и эта книга) иногда можно встретить комментарии, неприемлемые в профессиональных программах.

Оценить качество комментариев достаточно легко. Просто прочитайте их подряд (не заглядывая в код). Этого должно быть достаточно для получения представления не только о задаче, которую выполняет программа, но и об этапах ее выполнения. Например, вот комментарии к краткой программе на языке BASIC из Приложения Е:

```
' Программа на языке BASIC, предназначенная для вычисления  
' среднего для максимум 100 чисел  
' Среднее скольких чисел вы хотите вычислить?  
' Продолжим, если введенное число принадлежит к интервалу от 1 до 100  
' Цикл, складывающий числа для усреднения  
' Ввод следующего числа  
' Прибавление этого числа к имеющейся сумме  
' Вычисление среднего значения  
' Вывод результата
```

Читаемость

Читаемость кода также является субъективным понятием. Читаемый для одного пользователя текст может оказаться совершенно непонятным для другого. В действительности можно утверждать только одно: для автора программа будет более понятна, чем для всех остальных. Это желательно помнить, приступая к ее написанию. Разумеется, предполагается, что вы хотите сделать свое творчество понятным для других пользователей.

Наиболее пагубно сказывается на читаемости программ имеющий дурную репутацию оператор GOTO, в различных вариациях присутствующий во многих язы-

ках программирования, в том числе и в VBA. Я не собираюсь настраивать вас против этого оператора, просто с его помощью будет проиллюстрирован принцип хорошего программирования.

Функция оператора GOTO проста — он меняет точку выполнения программы. Например, нижеприведенный код на языке BASIC просит пользователя ввести положительное число. При вводе отрицательного числа оператор GOTO возвращает вас на первую строку программы (метку TryAgain), что приводит к повторному выполнению всех операций. Другими словами, программа будет повторяться до ввода положительного числа:

```
TryAgain:
INPUT "Введите положительное число: ", X
IF x <= 0 THEN GOTO TryAgain
```

Вышеприведенный код не является иллюстрацией хорошего стиля программирования, но, по крайней мере, он читабелен. А вот пример кода, разобраться в котором намного сложнее:

```
TryAgain:
INPUT "Введите число от 1 до 100: ", X
IF X > 100 THEN GOTO TooLarge
IF X <= 0 THEN GOTO TooSmall
PRINT "Вы ввели число: ", X
GOTO Done
TooLarge:
PRINT "Вы ввели слишком большое число"
GOTO TryAgain
TooSmall:
PRINT "Вы ввели слишком маленькое число"
GOTO TryAgain
Done:
END
```

Подобные примеры кода иногда называют «макаронной программой», так как пользователю требуется постоянно прыгать по программе, меняя точки выполнения. Только представьте, как будет выглядеть написанная в подобном стиле программа длиной в тысячу строк! Вот пример более читаемого кода, хотя это все еще не лучший из возможных вариантов:

```
TryAgain:
INPUT "Введите число от 1 до 100: ", X
IF X > 100 THEN
    PRINT "Вы ввели слишком большое число"
    GOTO TryAgain
ELSEIF x <= 0 THEN
    PRINT "Вы ввели слишком маленькое число"
    GOTO TryAgain
END IF
PRINT "Вы ввели число: ", X
END
```

А вот код, выполняющий ту же самую задачу без применения оператора goto. Без сомнения, большинством программистов подобный стиль будет признан самым лучшим:

```
DO
    INPUT "Введите число от 1 до 100: ", X
    IF X > 100 THEN
```



```

    PRINT "Вы ввели слишком большое число"
ELSEIF X <= 0 THEN
    PRINT "Вы ввели слишком маленькое число"
END IF
LOOP UNTIL X >= 1 AND X <= 100
PRINT "Вы ввели число: ", X
END

```

Иногда читаемость кода становится жертвой программистов, которые почему-то полагают, что их творчество выглядит особенно элегантно, хотя на самом деле оно сложно читается и подвержено ошибкам. Особенно это касается программирования на языке C. В качестве простого примера рассмотрим три строки кода на этом языке:

```

X = X + 1
X = X + I
i = i - 1

```

Первая строка увеличивает значение переменной X на единицу, вторая — складывает значения переменных X и i, а третья — уменьшает значение переменной i на единицу. Это хорошо читаемый код, но его можно записать и в другой форме:

```

X = ++X+i--;

```

Некоторые программисты могут счесть подобный код примером разумного подхода, но я считаю его вредным. Более дальновидно предпочесть читаемость краткости и элегантности.

Модульность

Другим важным моментом, имеющим отношение к читаемости, является модульность программы. В эпоху зарождения программирования на персональных компьютерах (для этого использовался язык BASIC) большинство программ представляли собой единый блок, иногда состоящий из сотен и даже тысяч строк. Следить за содержанием подобной программы было непросто, особенно через несколько месяцев после завершения работы над ней. Кроме того, часто такие программы содержали одинаковые сегменты, что являлось пустой тратой времени и места.

Проиллюстрируем вышесказанное на примере программы на языке BASIC. Номера строк добавлены, чтобы вам было проще следить за происходящим. Не стоит расстраиваться, если вы не поймете смысла отдельных строк. Это не мешает следить за обсуждением.

```

10 ' Программа, меняющая порядок букв в вашем имени
20 ' Обработка имени
30 INPUT "Введите ваше имя: ", name$
40 reverse$ = ""
50 FOR I = LEN(name$) TO 1 STEP -1
60     reverse$ = reverse$ + MID$(name$, i, 1)
70 NEXT I
80 PRINT "Имя наоборот: " + reverse$

90 ' Обработка отчества
100 INPUT "Введите ваше отчество: ", name$
110 reverse$ = ""
120 FOR I = LEN(name$) TO 1 STEP -1

```



```
130 reverse$ = reverse$ + MID$(name$, i, 1)
140 NEXT I
150 PRINT "Отчество наоборот: " + reverse$

160 ' Обработка фамилии
170 INPUT "Введите вашу фамилию: ", name$
180 reverse$ = ""
190 FOR I = LEN(name$) TO 1 STEP -1
200 reverse$ = reverse$ + MID$(name$, i, 1)
210 NEXT I
220 PRINT "Фамилия наоборот: " + reverse$
```

Теперь обратите внимание на то, что строки 40–70, 110–140 и 180–210 (выделенные полужирным стилем) совершенно идентичны. Это пустая трата места. Имеет смысл поместить код, изменяющий порядок букв, в отдельный модуль, который будет вызываться три раза, как в нижеприведенном примере:

```
'Программа, изменяющая порядок букв в имени
DECLARE FUNCTION Reverse$ (name$)

'Обработка имени
INPUT "Введите имя: ", name$
PRINT "Имя наоборот: " +Reverse$(name$)

'Обработка отчества
INPUT "Введите отчество: ", name$
PRINT "Отчество наоборот: " +Reverse$(name$)

'Обработка фамилии
INPUT "Введите фамилию: ", name$
PRINT "Фамилия наоборот: " +Reverse$(name$)
```

Отдельный модуль кода, изменяющий порядок букв, выглядит следующим образом:

```
'Изменение порядка букв
FUNCTION Reverse$(aname$)
Temp$ = ""
FOR i = LEN(aname$) TO 1 STEP -1
Temp$ = Temp$ + MID$(aname$, i, 1)
NEXT I
Reverse$ = Temp$
END FUNCTION
```

Конечно, в данном примере экономия места является не очень значительной, но представьте, что случится, если повторяющаяся процедура занимает сотню строк кода и фигурирует в программе много раз. В этом случае разбиение на модули может уменьшить код на тысячи строк.

Программирование в виде модулей имеет еще одно преимущество. Если в другой программе потребуется поменять порядок букв в словах, достаточно будет добавить в нее уже существующий модуль. Вам не потребуется писать новый код. Профессиональные программисты часто пользуются библиотеками модулей, которые можно включить в программу.

Важность модульного подхода трудно переоценить. К счастью, вы скоро убедитесь в том, что язык VBA позволяет легко создавать модульные программы.

Вообще говоря, существуют две группы модулей: функции и подпрограммы. Разница между ними состоит в том, что функция возвращает определенное значение, а подпрограмма нет. Скажем, функция Reverse, описанная в предыдущем при-

мере, возвращает реверсированную строку. С другой стороны, нижеприведенный модуль выполняет определенные действия, но не возвращает никакого значения. Он просто останавливает работу на несколько секунд:

```
SUB delay (sec)
  'Введите текущее время
  StartTime = TIMER
  ' Создание цикла задержки на sec секунд
  DO
  LOOP UNTIL TIMER - StartTime > sec
END SUB
```

Функции и подпрограммы повсеместно встречаются в современных кодах. Они известны под общим названием *процедуры*.

3

Редактор Visual Basic: часть I

Первым шагом к программированию на языке VBA в Excel является знакомство со средой, в которой оно происходит. Каждое из основных приложений Microsoft Office имеет собственную интегрированную среду разработки (IDE — Integrated Development Environment). Компания Microsoft называет эту среду редактором Visual Basic.

В этой и последующей главах будут описаны основные компоненты IDE Excel. Я понимаю, что вам не терпится приступить к написанию программ, но сначала требуется получить информацию о редакторе, с которым вам предстоит работать. Впрочем, никто не мешает быстро просмотреть эти две главы, а потом возвращаться к ним по мере необходимости.

В пакете Office 97 интегрированная среда разработки приложений Word, Excel и PowerPoint имеет один и тот же вид, показанный на рис. 3.1. Начиная с пакета Office 2000 так же выглядит среда разработки для Microsoft Access. Для запуска редактора Visual Basic воспользуйтесь командой Tools ► Macros ► Visual Basic Editor (Сервис ► Макрос ► Редактор Visual Basic) главного меню или нажмите комбинацию клавиш Alt + F11.

Рассмотрим более подробно некоторые компоненты этого редактора.

Окно проекта

Окно проекта расположено в верхнем левом углу клиентской области (под панелью инструментов). Оно показано на рис. 3.2.

Это окно содержит иерархический список элементов проекта, напоминающий дерево папок, которое вы привыкли наблюдать в левой части приложения Windows Explorer. Элементы окна проекта называются *узлами (nodes)*. Узлы верхнего уровня (на рис. 3.2 их два, и они выделены полужирным шрифтом) представляют собой проекты, открытые в данный момент. Чтобы раскрыть дерево иерархии проекта или свернуть его, достаточно щелкнуть на квадратике, расположенном слева от его имени. Обратите внимание на то, что существует один проект для каждой книги, открытой в данный момент.

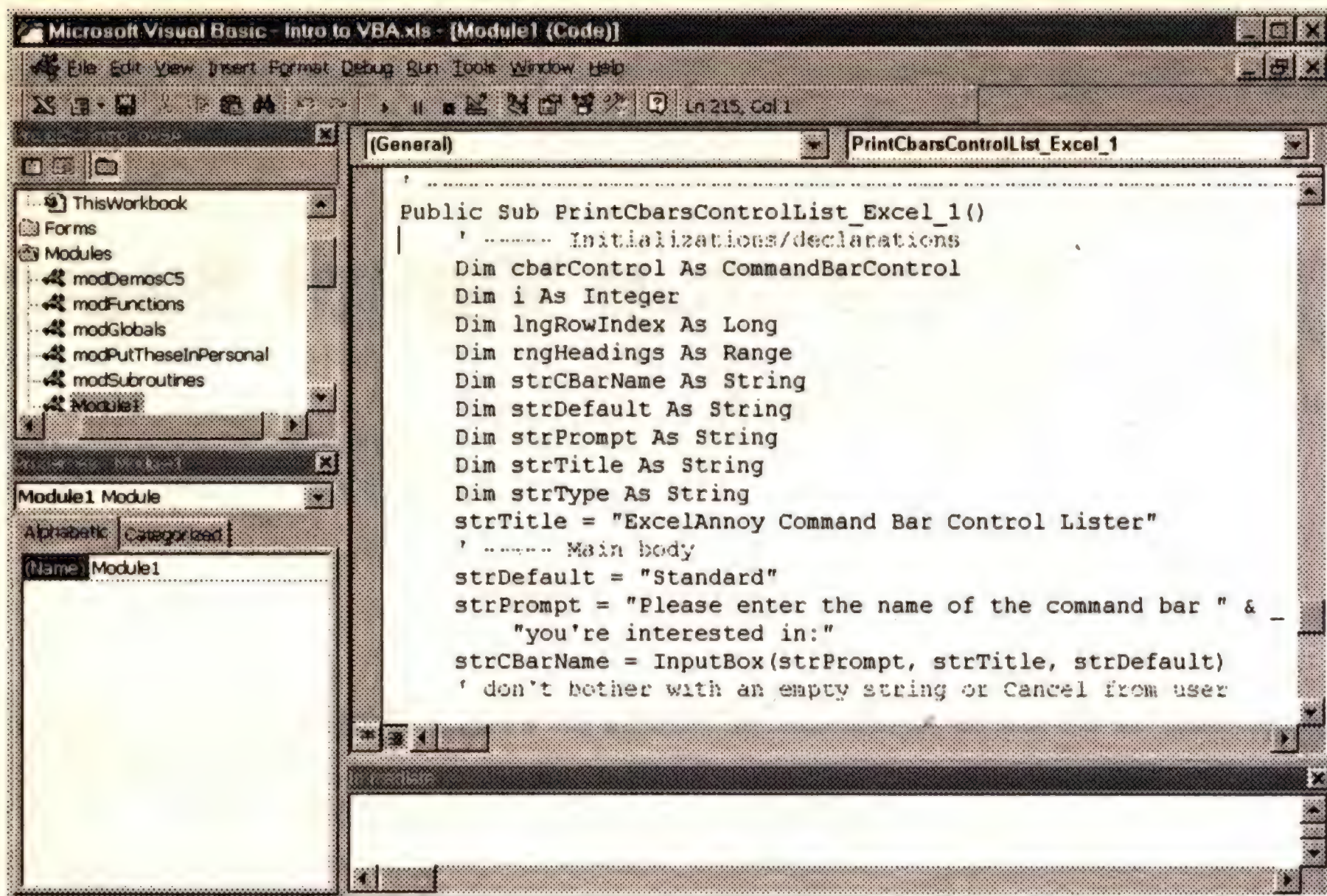


Рис. 3.1. Интегрированная среда разработки Excel VBA

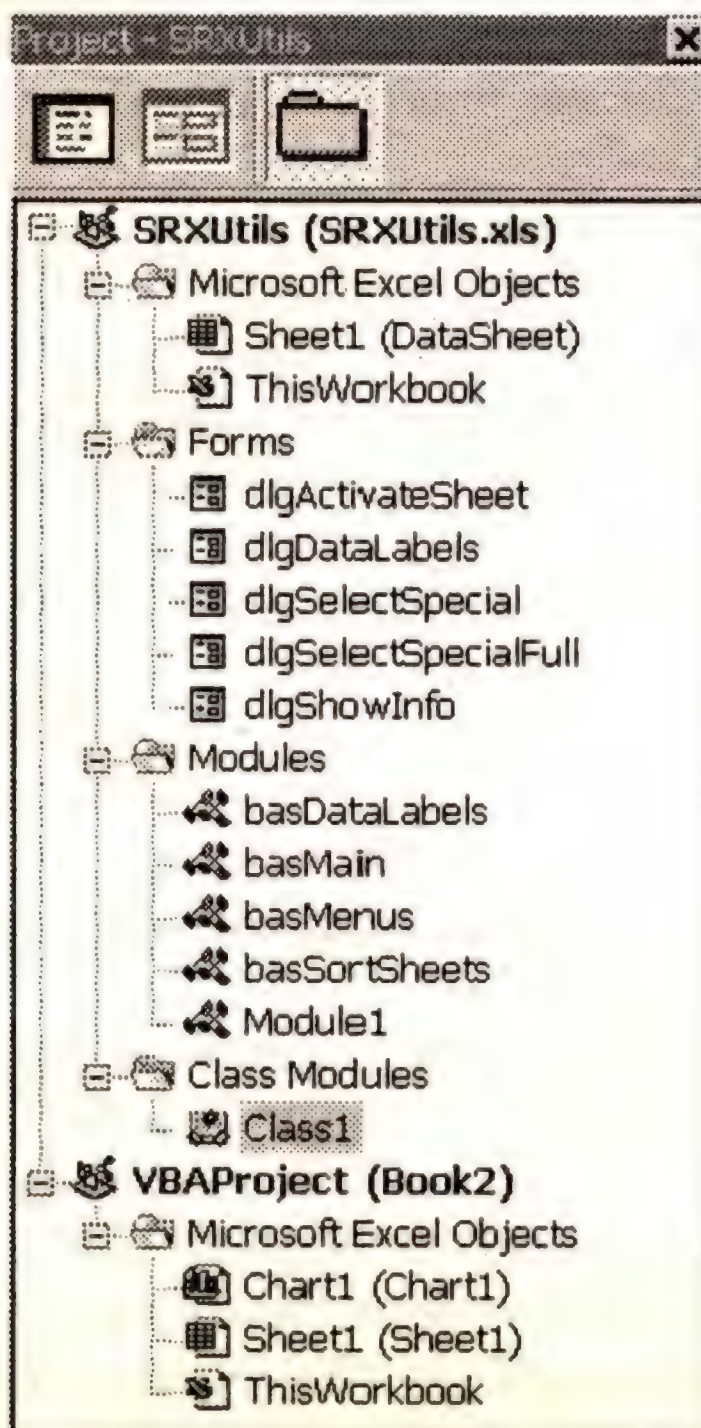


Рис. 3.2. Окно проекта

Имена проектов

Каждый проект имеет собственное имя, которое пользователь может изменять по своему усмотрению. По умолчанию каждому проекту присваивается имя `VBAProject`. Узлам верхнего уровня назначаются имена вида `ProjectName (WorkbookName)`. Здесь `ProjectName` — имя проекта, а `WorkbookName` — имя книги.

Содержимое проекта

На втором уровне, как показано на рис. 3.2, находятся следующие узлы:

- объекты Microsoft Excel;
- формы;
- модули;
- модули классов.

Под объектным узлом Microsoft Excel располагаются объектные узлы `Worksheet` и `Chartsheet`, а также специальный узел `ThisWorkbook`, который представляет саму книгу. Эти узлы обеспечивают доступ к окнам с кодом для каждого из объектов, в которых вы можете писать собственный код.

Узел `Form` включает в себя узлы для всех форм проекта. Формами в данном случае называются нестандартные диалоговые окна. Более подробную информацию о них вы получите чуть позднее. Соответственно, под узлами `Modules` и `Classes` располагаются узлы, соответствующие всем модулям и модулям классов проекта. О них мы тоже поговорим чуть позже.

Окно проекта предназначено для перемещения между различными узлами. Листы и формы состоят из двух компонентов — видимый пользователю компонент (собственно лист или диалоговое окно) и скрытый код. Щелчок правой кнопкой мыши на имени соответствующего узла вызывает меню, команды которого позволяют посмотреть как сам объект, так и его код. Стандартные модули и модули классов имеют только код, доступ к которому можно получить, дважды щелкнув на имени узла.

Рассмотрим подробно все компоненты проекта Excel.

Объект `ThisWorkbook`

Под каждым узлом объекта Microsoft Excel располагается узел с именем `ThisWorkbook`. Он представляет книгу проекта и ее код (также называемый модулем кода), который сохраняет события для книги. В этот компонент можно также поместить независимую процедуру, которая обычно входит в состав стандартных модулей. С ними вы познакомитесь немного позднее.

События инициируют запуск определенных процедур. С книгами Excel связано 19 событий. Подробно мы поговорим о них в главе 11, а пока я приведу несколько примеров:

- событие `Open` возникает при открытии книги;
- событие `BeforeClose` наступает в момент закрытия книги;
- событие `NewSheet` наступает при добавлении в книгу нового листа;
- событие `BeforePrint` наступает перед распечаткой книги или части ее содержимого.

Объекты Sheet

Под каждым узлом объекта Microsoft Excel также располагаются узлы для каждого листа книги. При этом лист содержит таблицу или диаграмму, и, кроме того, в его состав входит компонент, включающий код событий. В этот компонент можно также поместить независимые процедуры, которые обычно находятся в стандартном модуле.

В Excel существуют 7 событий для объекта Worksheet и 13 событий для объекта Chartsheet. Подробно вы познакомитесь с ними в главе 11.

Стандартные модули

Модули, которые также принято называть стандартными модулями, содержат процедуры (функции и подпрограммы). Это могут быть макросы, которые запускаются пользователем, или служебные программы, используемые другими программами. Обсуждение модульного программирования можно найти в предыдущей главе.

Модули классов

Модули классов содержат код, относящийся к нестандартным объектам. Как вы вскоре убедитесь, объектная модель Excel включает множество встроенных объектов (их число почти достигает 200), таких как книги, листы, диаграммы, шрифты и т. п. При этом существует возможность создавать нестандартные объекты и присваивать им определенные свойства. Это делается путем помещения соответствующего кода в модуль классов.

Однако разговор о создании нестандартных объектов выходит за рамки данной книги, поэтому вам не придется сталкиваться с модулями классов. Объектно-ориентированному программированию на языке Visual Basic посвящена моя другая книга *Concepts of Object-Oriented Programming with Visual Basic* (издательство Springer-Verlag, New York).

Объекты UserForm

Как вы знаете, Excel содержит множество встроенных диалоговых окон. При этом существует возможность создания ваших собственных диалоговых окон, которые называются формами, или UserForm. На рис. 3.3 показана среда разработки диалогового окна Select Special, упоминавшегося в главе 1.

Область, расположенная на рис. 3.3 справа, содержит диалоговое окно dlgSelectSpecial в режиме конструирования. Справа от него находится плавающее окно Toolbox (Элементы управления) со значками различных элементов управления.

Чтобы поместить элемент управления в диалоговое окно, перетащите нужный значок окна Toolbox (Элементы управления) и поменяйте размер появившегося прямоугольника. Прямоугольник будет заменен элементом управления такого же размера. Свойства объекта UserForm, а также свойства любых элементов управления можно менять в окне Properties, о котором пойдет речь в следующем разделе.

Кроме формы и элементов управления объект UserForm содержит код на языке VBA. Например, командной кнопке соответствует событие Click, наступающее после щелчка пользователя на кнопке. Если поместить такую кнопку в диалоговое окно, потребуется написать код, который будет запускаться при наступлении события Click. Без этого щелчок на кнопке не даст никакого результата. Ниже приве-

ден код для события Click кнопки Close из рис. 3.3. Обратите внимание на то, что свойство Name командной кнопки имеет значение cmdClose:

```
Private Sub cmdClose_Click()  
    Unload Me  
End Sub
```

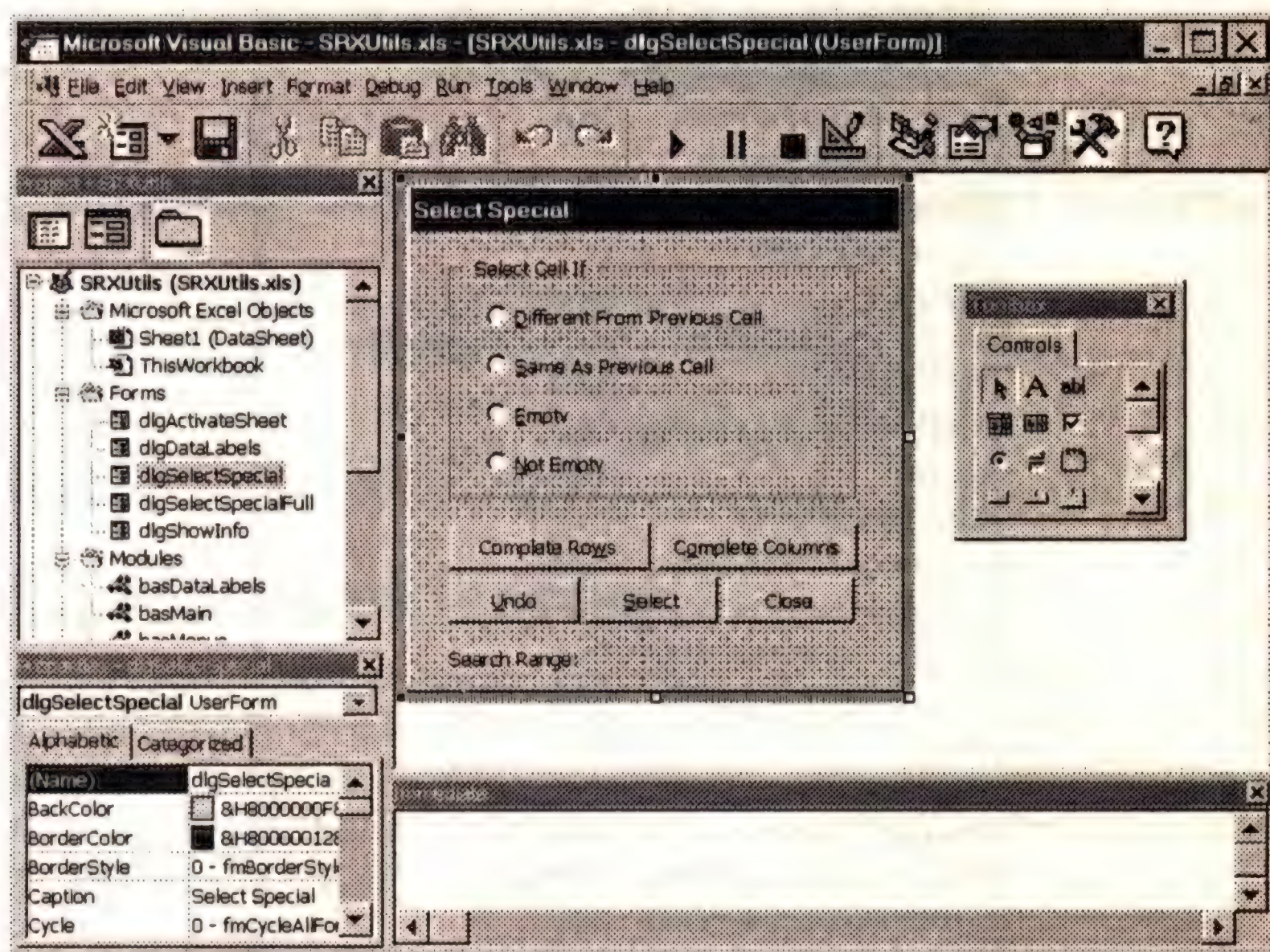


Рис. 3.3. Создание нестандартного диалогового окна

Этот код закрывает диалоговое окно.

Кроме кода события для диалогового окна и его элементов управления в объект UserForm можно включить вспомогательные процедуры.

Не беспокойтесь, если пока что вам не совсем понятно, о чем идет речь. Созданию нестандартных диалоговых окон посвящена целая глава с примерами из жизни.

Окно Properties

Окно Properties, показанное на рис. 3.1, дает доступ к изменению свойств объекта.

При выделении стандартного модуля единственным свойством, которое появится в окне Properties, будет имя этого модуля. Однако выделение книги, листа или нестандартного диалогового окна приводит к появлению многочисленных свойств, как показано на рис. 3.4.

Окно Properties можно использовать для изменения свойств объекта даже при отсутствии выполняемого кода, то есть в период проектирования. Обратите внимание на то, что некоторые свойства предназначены только для чтения, то есть лишены возможности их редактировать. Большинство свойств можно менять как в период проектирования, так и во время выполнения кода. Однако существуют

свойства, которые допускают редактирование или во время проектирования, или во время выполнения кода. Последние обычно не появляются в окне Properties.

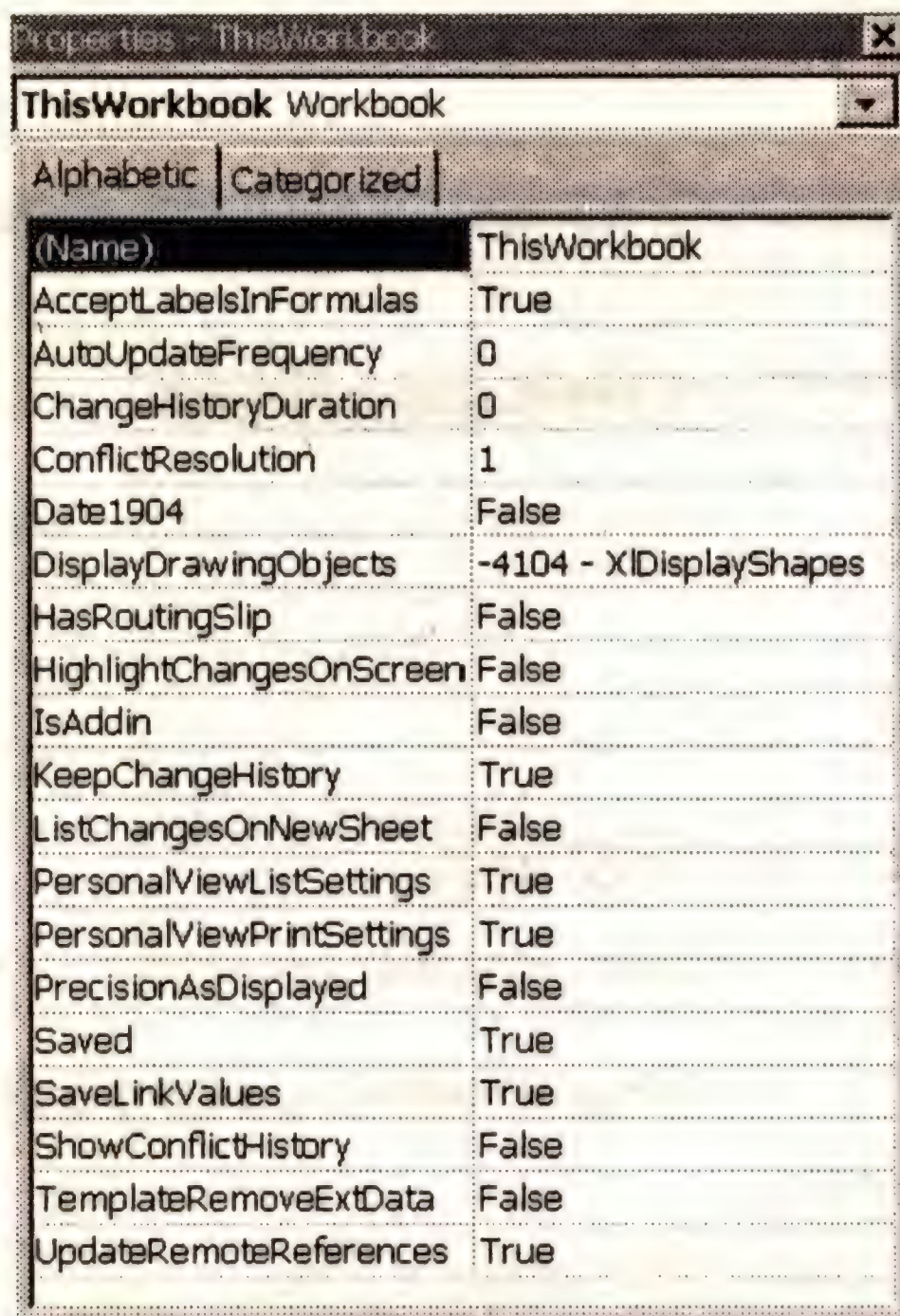


Рис. 3.4. Окно Properties

Окно Code

В окне Code демонстрируется код элемента, выделенного в окне Project. Чтобы сделать его видимым, выберите в меню View редактора VBA команду Code или нажмите клавишу F7. Если у объекта отсутствует видимый компонент, достаточно дважды щелкнуть на его имени в окне Project.

Представления полного модуля и процедуры

В общем случае модуль кода (стандартный, модуль классов или UserForm) содержит несколько процедур. Представление, в котором видна в данный момент времени только одна процедура, называется *представлением процедуры (procedure view)*. Чтобы увидеть все процедуры сразу, разделенные горизонтальными линиями, требуется перейти в *представление полного модуля (full-module view)*. Оба представления имеют свои преимущества и недостатки, и, скорее всего, вам предстоит работать с ними обоими. К сожалению, возможность выбора представления в меню не предусмотрена. Для перехода от одного представления к другому требуется

щелкнуть на одной из маленьких кнопок, расположенных в нижнем левом углу окна Code. Выбор представления, показываемого по умолчанию, осуществляется на вкладке Editor диалогового окна Options.

К сожалению, по умолчанию для отображения информации модуля кода выбран шрифт Courier, в некоторых случаях затрудняющий чтение. В этом случае можно выбрать в меню Tools команду Options и в раскрывающемся списке Font на вкладке Editor Format указать более читаемый шрифт, например FixedSys.

Раскрывающиеся списки Object и Procedure

В верхней части окна Code находятся два раскрывающихся списка, как показано на рис. 3.1. Левый содержит список всех объектов (например, диалоговых окон или элементов управления), связанных с данным проектом, а правый — список всех процедур, связанных с выбранным объектом. Вид этих раскрывающихся списков меняется в зависимости от типа объекта, выделенного в окне Project.

Объекты Workbook и Sheet

Когда в окне Project выделена книга или лист, раскрывающийся список Object содержит только два элемента — общую область (для общих процедур) и рассматриваемый объект (для книги, листа или диаграммы). После выбора второго варианта в раскрывающемся списке Procedure появятся пустые оболочки кода для событий, имеющих отношение к данному объекту. Пример показан на рис. 3.5.

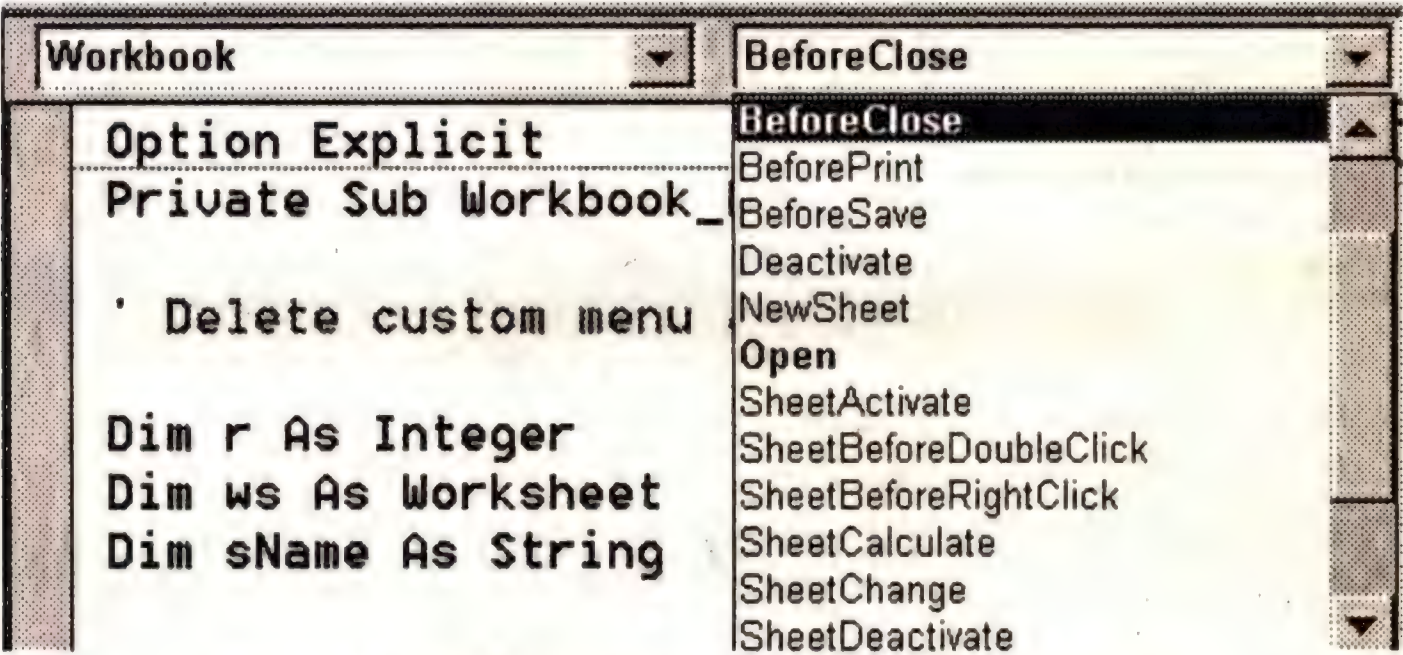


Рис. 3.5. Список событий для объекта Workbook

Если, к примеру, выбрать в раскрывающемся списке Procedure событие BeforeClose, появится следующий код:

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
End Sub
```

Стандартный модуль

При выборе в окне Project стандартного модуля раскрывающийся список Object будет содержать только элемент General. В раскрывающемся списке Procedure находится список всех процедур, написанных для данного модуля, как показано на рис. 3.6.

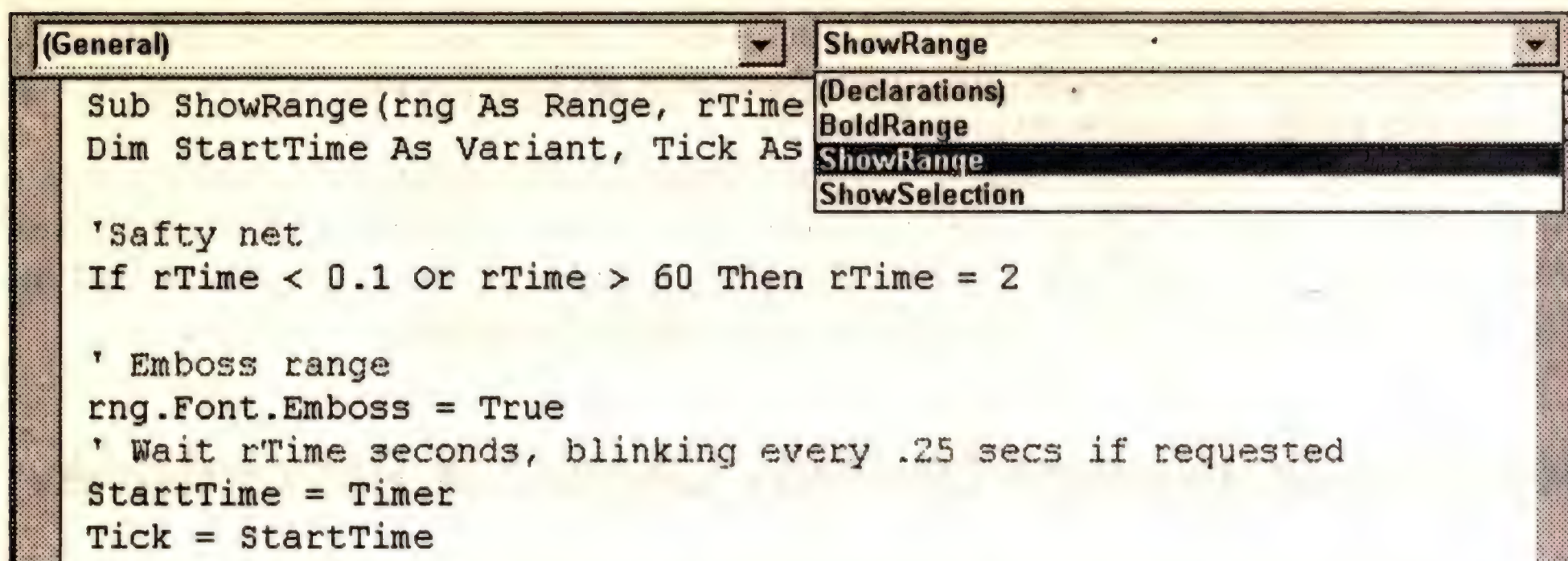


Рис. 3.6. Содержимое раскрывающегося списка Procedure

В раздел Declarations помещаются описания переменных уровня модуля. Их можно использовать в любой процедуре модуля, для которой она была определена. Подробно мы поговорим об этих переменных в главе 5.

Объект UserForm

При выборе в окне Project объекта UserForm раскрывающийся список Object будет содержать список всех объектов, составляющих нестандартную форму. Например, на рис. 3.7 показан вид списка Object (Объект) для диалогового окна, изображенного на рис. 3.3.

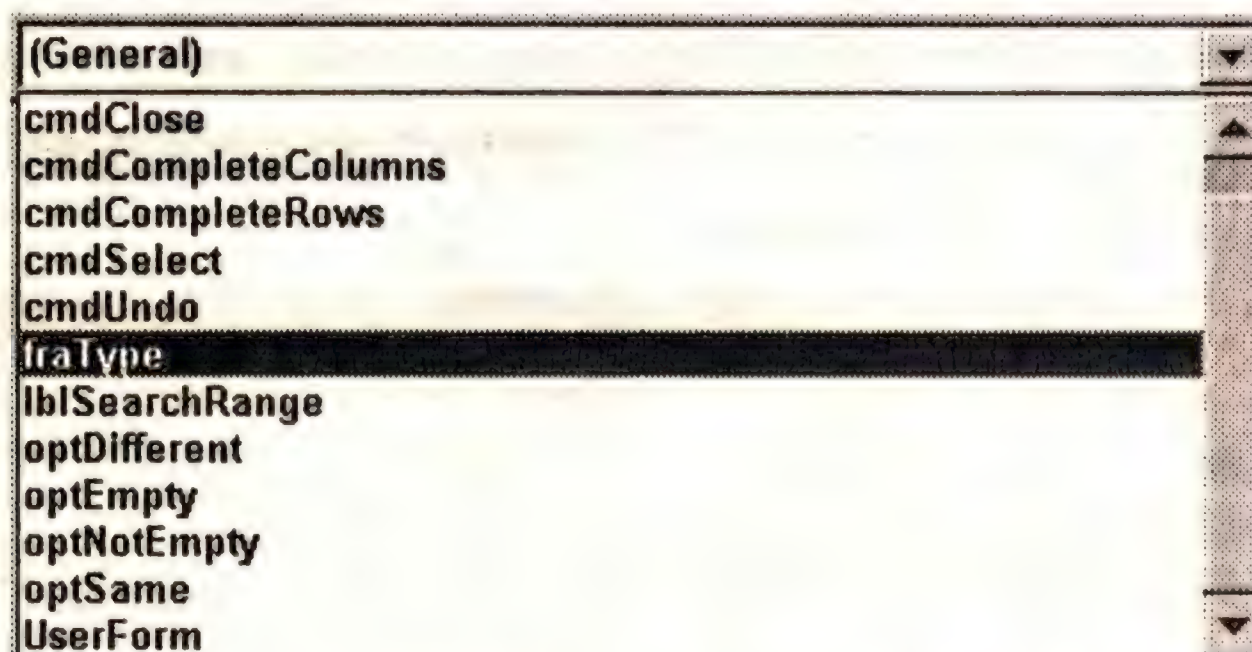


Рис. 3.7. Вид раскрывающегося списка Object для диалогового окна, изображенного на рис. 3.3

Обратите внимание на наличие элементов для различных командных кнопок (например, cmdClose), прочих элементов управления и даже для самого объекта UserForm.

На рис. 3.8 показан вид раскрывающегося списка Procedure после выбора в раскрывающемся списке Object варианта cmdClose. Этот список содержит названия 13 различных событий, связанных с указанной командной кнопкой.

К примеру, при выборе события Click в окне Code (Код) окажется следующая процедура:

```

Private Sub cmdClose_Click()
End Sub
  
```

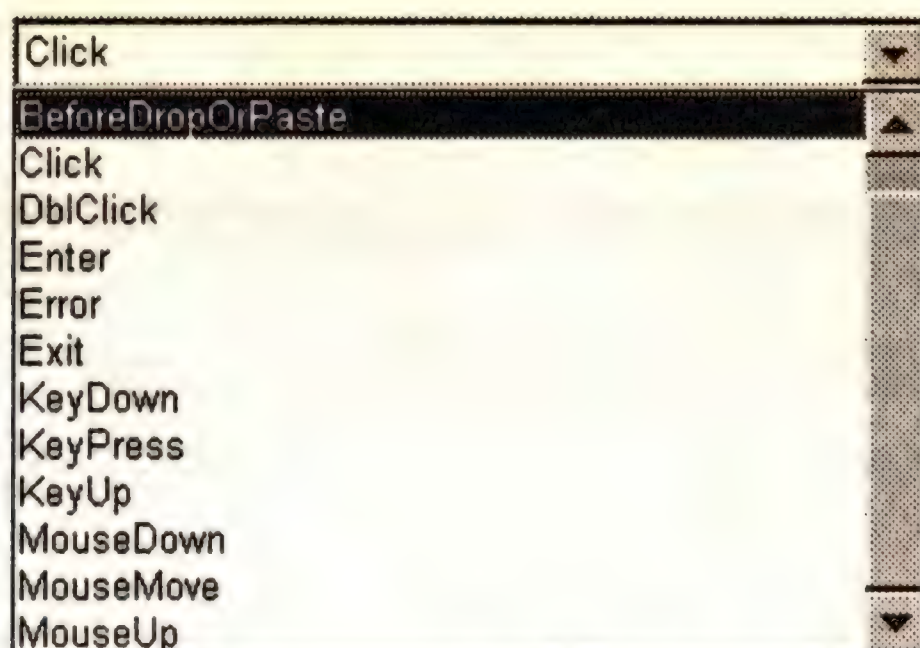



Рис. 3.8. Вид раскрывающегося списка Procedure для объекта cmdClose

Окно Immediate

Окно Immediate, показанное на рис. 3.1, имеет две основные функции. Во-первых, можно послать в него выходные данные с помощью команды `Debug.Print`. Например, следующий код выведет информацию из ячейки A1 активного листа в окно Immediate:

```
Debug.Print ActiveSheet.Range("A1").Text
```

Это дает превосходную возможность экспериментировать с различными частями кода.

Второй основной функцией данного окна является выполнение команд. К примеру, достаточно выделить текст в активном документе, ввести в окно Immediate команду, показанную на рис. 3.9, нажать клавишу Enter, и текст будет выделен полужирным шрифтом.

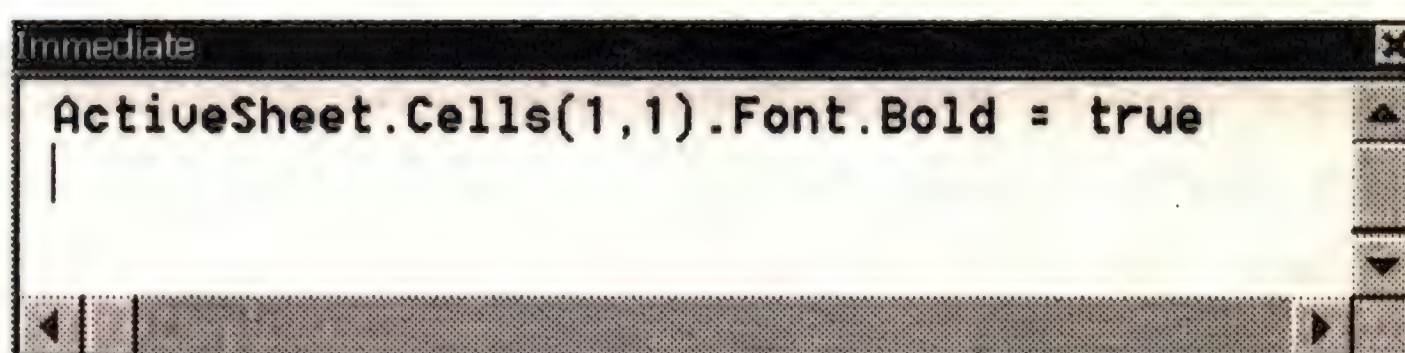


Рис. 3.9. Окно Immediate

Окно Immediate является просто бесценным инструментом для отладки программ, и, без сомнения, вам придется использовать его достаточно часто.

Расположение окон

Чтобы получить дополнительное место для написания кода, можно закрыть окна Properties, Project и Immediate. С другой стороны, если у вас достаточно большой монитор, можно разделить экран, как показано на рис. 3.10, чтобы одновременно видеть и среду разработки Excel VBA и активную книгу. Это позволит наблюдать

результат выполнения кода прямо в книге. Для перехода от Excel к среде разработки используется комбинация клавиш Alt + F11.

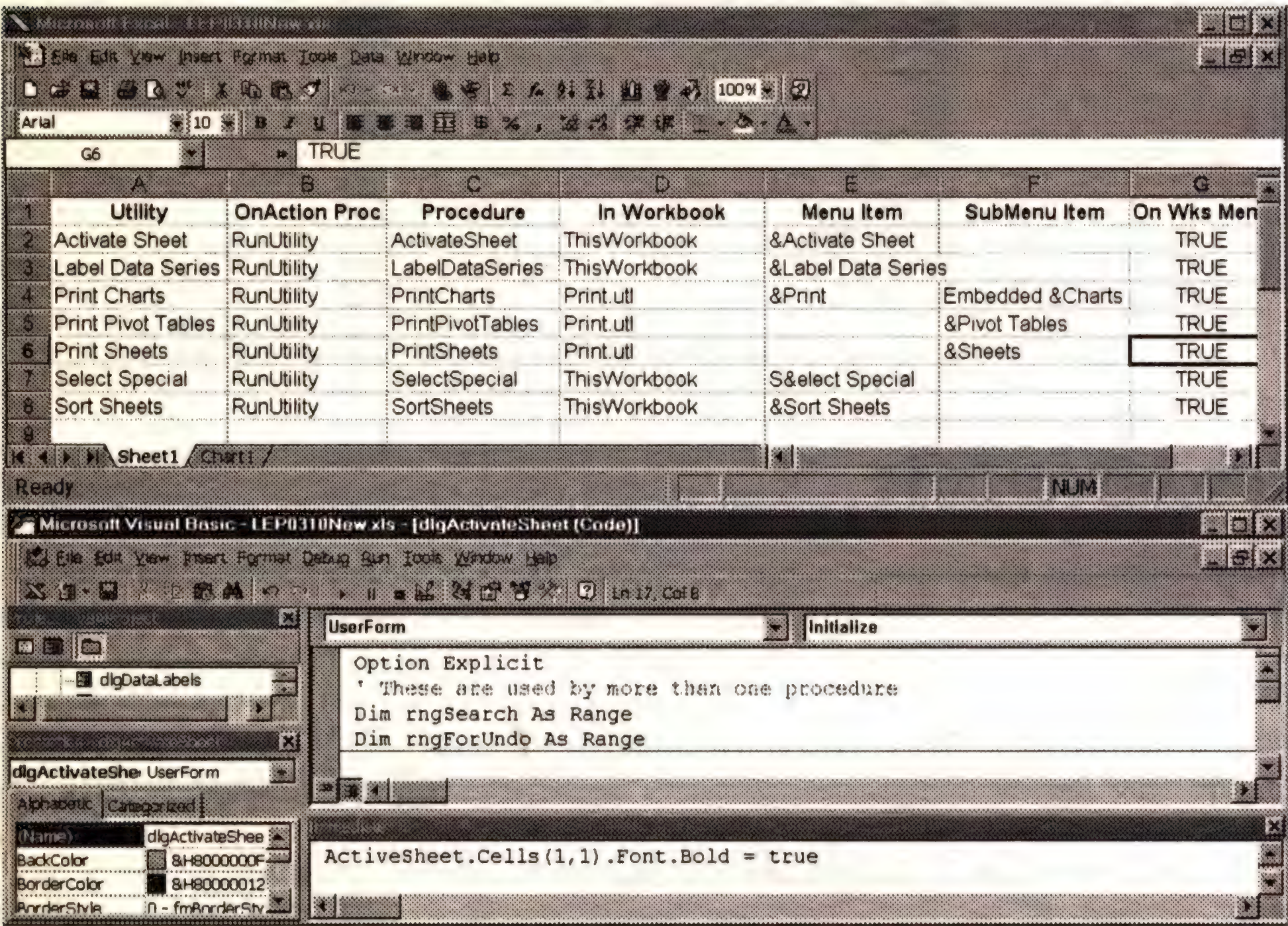


Рис. 3.10. Вариант одновременного размещения на экране книги Excel и среды разработки Excel VBA

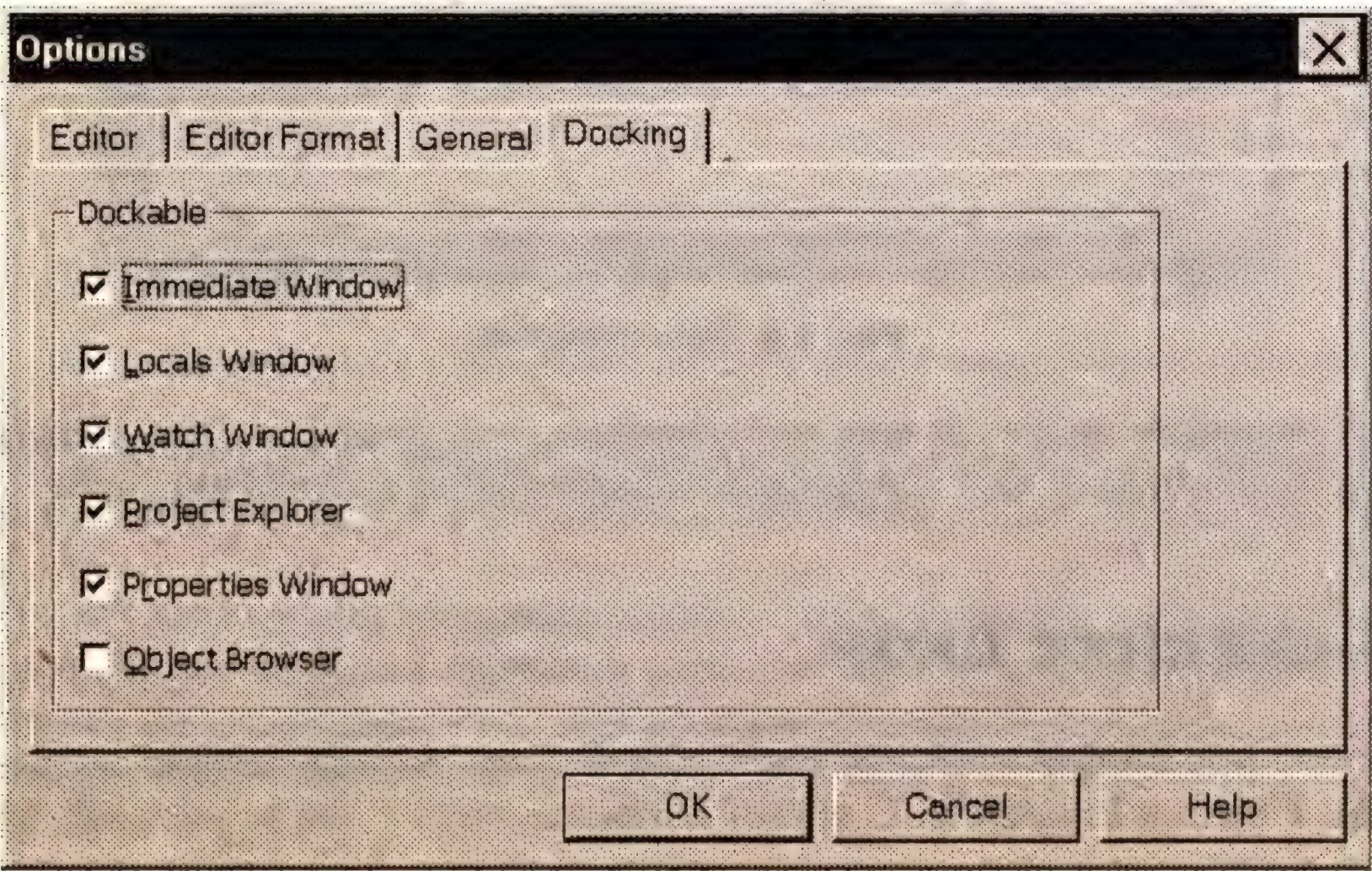


Рис. 3.11. Параметры пристыковки

Пристыковка

Большинство окон интегрированной среды разработки (в том числе окна Properties, Project и Immediate) могут как находиться в плавающем состоянии, так и быть пристыкованными к одной из сторон экрана. Состояние окон задается на вкладке Docking диалогового окна Options, как показано на рис. 3.11.

Перемещение пристыкованных окон осуществляется только между набором закрепленных позиций, в то время как плавающие окна располагаются в любом месте экрана по вашему желанию.

4

Редактор Visual Basic: часть II

В этой главе мы завершим обсуждение свойств редактора Visual Basic. Еще раз напомним читателям, что на данный момент эту главу можно просто просмотреть, а потом возвращаться к ней по мере необходимости.

Управление интегрированной средой разработки

Если вы вместо мыши предпочитаете использовать клавиатуру (как это делаю я), потребуется выучить клавиатурные комбинации. Перечислим некоторые из них.

Общее управление

Следующие клавиатурные комбинации используются для управления интегрированной средой разработки:

- F7 — переход к окну Code;
- F4 — переход к окну Properties;
- Ctrl + R — переход к окну Project;
- Ctrl + G — переход к окну Immediate;
- Alt + F11 — переход от Excel к интегрированной среде разработки.

Управление окном Code в период проектирования

Для работы в окне Code потребуются следующие клавиатурные комбинации:

- F1 — информация об объекте, расположенном под курсором;
- Shift + F2 — переход к описанию элемента, расположенного под курсором. Если курсор установлен на вызов процедуры, эта комбинация клавиш приведет к появлению ее описания;
- Ctrl + Shift + F2 — возвращает курсор в предыдущую точку редактирования.

Отладка кода

Следующие клавиатурные комбинации пригодятся вам при отладке кода. О ней мы подробно поговорим чуть позднее.

- F8 — шаг с заходом. Приводит к пошаговому выполнению подпрограммы или функции.
- Shift + F8 — шаг с обходом. Приводит к моментальному выполнению подпрограммы (функции) целиком.
- Ctrl + Shift + F8 — шаг с выходом. Выполнение остановится на инструкции, следующей за вызовом.
- Ctrl + F8 — выполнение кода до места расположения курсора.
- F5 — выполнение кода.
- Ctrl + Break — прерывание выполнения кода.
- Shift + F9 — появляется диалоговое окно Quick Watch, позволяющее наблюдать за изменением определенной переменной в процессе выполнения кода.
- F9 — точка останова. Будет создано место, в котором нормальное выполнение программы приостанавливается. Это полезно, если требуется быстро выполнить код до определенной инструкции, а затем перейти в пошаговый режим отладки кода.
- Ctrl + Shift + F9 — снять все точки останова.

Закладки

Вы можете установить в определенных точках кода закладки. Они позволяют легко вернуться в нужное место программы. Для включения закладок, а также для перехода между ними используется меню, вызываемое командой Edit ► Bookmark. Наличие закладок отмечается небольшими голубыми прямоугольниками, расположенными слева от кода.

Справочная система

Скорее всего, в процессе программирования вам не раз придется прибегнуть к справочной системе VBA. Проще всего получить информацию о любом из элементов, поместив на него курсор и нажав клавишу F1. Это допустимо не только для ключевых слов языка VBA, но и для ряда элементов интегрированной среды разработки.

Имейте в виду, что компания Microsoft предоставила множество вспомогательных файлов по работе с приложением Excel языку VBA и объектной модели Excel. Несмотря на разумность такого подхода, иногда система помощи отказывается выдать правильный файл после нажатия клавиши F1. Я не нашел простого решения данной проблемы и поэтому просто предпочитаю закрыть Excel и редактор Visual Basic и затем запустить их снова.

Помните, что при стандартной установке пакета Microsoft Office справочные файлы по языку VBA для различных приложений не устанавливаются. Следовательно, вам может потребоваться повторный запуск программы по установке Office. Не путайте справочные пособия по Excel и по Excel VBA.

Создание процедур

Существуют два способа создания новых процедур (то есть подпрограмм и функций) внутри модулей кода. Во-первых, после выбора нужного проекта в окне Project можно воспользоваться командой Procedure меню Insert. Это приведет к появлению диалогового окна, показанного на рис. 4.1.

Введите в поле Name имя процедуры и установите переключатель Type в положение Sub или Function. Установка переключателя в положение Property используется при создании объектов в модуле классов.

Вторым способом создания процедур является ввод строки:

Sub Имя_подпрограммы

или

Function Имя_функции

в окно Code. Для завершения процедуры используются операторы End Sub и End Function соответственно. После нажатия клавиши Enter строки кода будут перемещены на новое место, что приведет к созданию новой процедуры.

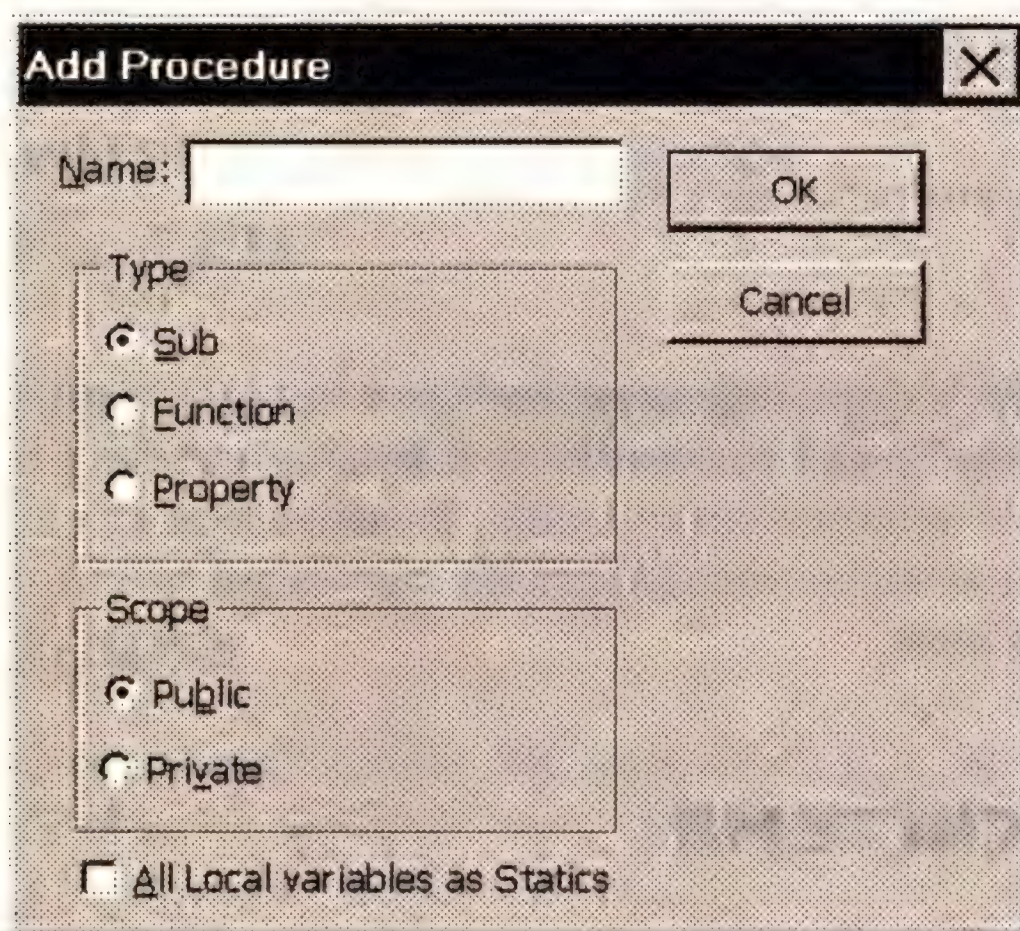


Рис. 4.1. Диалоговое окно Add Procedure

Режимы выполнения, проектирования и прерывания

Интегрированная среда разработки VBA работает в одном из трех режимов: выполнения, прерывания и проектирования. Последний режим дает возможность писать код и конструировать формы.

Режим выполнения используется при запуске процедур. Для выполнения процедуры достаточно нажать клавишу F5 или выбрать в меню Run команду Run Sub/UserForm. Если создается впечатление, что процедура зависла, ее можно прервать с помощью комбинации клавиш Ctrl + Break.

Режим прерывания возникает при остановке процедуры из-за ошибки в коде или преднамеренных действий с вашей стороны. О них мы поговорим чуть позднее. При наличии ошибок приложение прерывает выполнение программы и вызывает диалоговое окно, показанное на рис. 4.2.

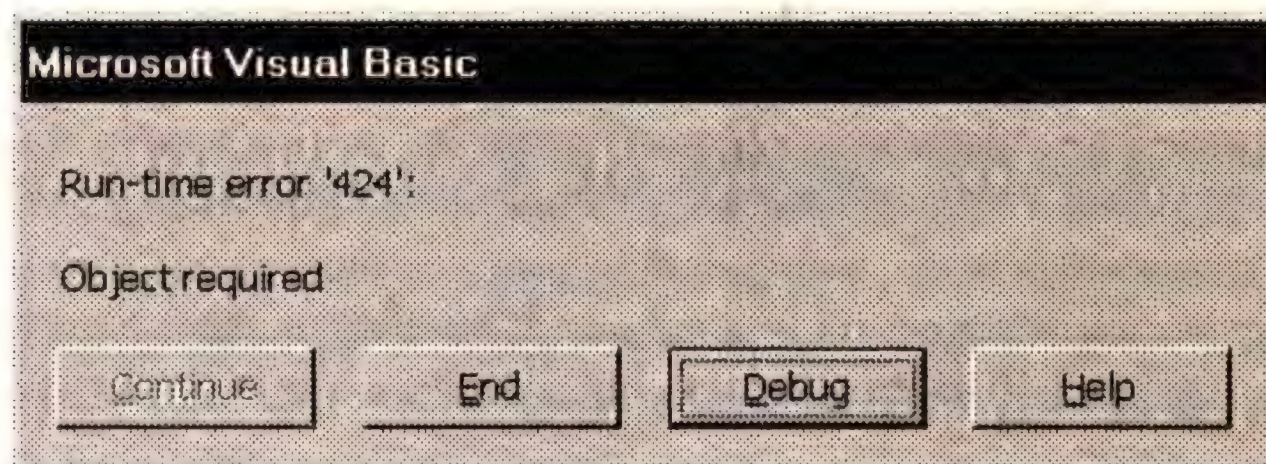


Рис. 4.2. Диалоговое окно с сообщением об ошибке

Диалоговые окна с сообщениями об ошибке дают несколько возможностей — завершить процедуру, получить справку об ошибке или войти в режим прерывания для отладки кода. В последнем случае приложение останавливает выполнение программы на строке с ошибкой и выделяет эту строку желтым цветом. Процесс отладки мы обсудим немного позже.

Существует еще ряд способов входа в режим прерывания:

- Комбинация клавиш Ctrl + Break и выбор в появившемся меню команды Debug.
- Включение в код оператора Stop, который заставляет приложение перейти в режим прерывания.
- Включение *точки останова (breakpoint)* в существующий код. Для этого достаточно поместить курсор в нужную строку кода и нажать клавишу F9 или воспользоваться командой Toggle Breakpoint меню Debug. Слева от выделенной строки появится красная точка. Внутри одной процедуры можно поместить несколько точек останова. Обычно это более предпочтительный способ, чем использование оператора Stop, так как точки останова автоматически удаляются, когда вы закрываете редактор Visual Basic, в то время как оператор требуется удалять вручную.
- Использование оператора Watch, который заставляет приложение войти в режим прерывания при соблюдении определенного условия.

Для выхода из режима прерывания используйте команду Reset меню Run.

Обратите внимание на то, что в строке заголовка редактора VBA указывается, в каком режиме вы работаете в данный момент. Режиму выполнения соответствует слово [running], а режиму прерывания — слово [break].

Ошибки

На компьютерном жаргоне ошибки называются *багами (bug)*. В переводе с английского это слово означает «жук». Этот термин появился из-за ошибки в работе одного из первых крупных компьютеров Mark I (примерно в 1944 году), причиной которого была попавшая в аппаратное обеспечение моль. Этот компьютер состоял

из 750 000 блоков, имел длину 51 фут и весил свыше пяти тонн. Только представьте себе такого монстра в роли настольного компьютера! Выполнение одной команды занимало примерно шесть секунд, в то время как современный компьютер с процессором Pentium за одну секунду выполняет свыше 200 миллионов команд.

Ошибки, возникающие во время проектирования

Как легко догадаться по заголовку данного раздела, речь в нем пойдет об ошибках, возникающих в процессе написания кода. Можно заставить редактор Visual Basic останавливать написание кода, когда вы делаете синтаксическую ошибку. Включение данной функции осуществляется на вкладке Editor диалогового окна Options, как показано на рис. 4.3. Я настоятельно рекомендую вам сделать это.

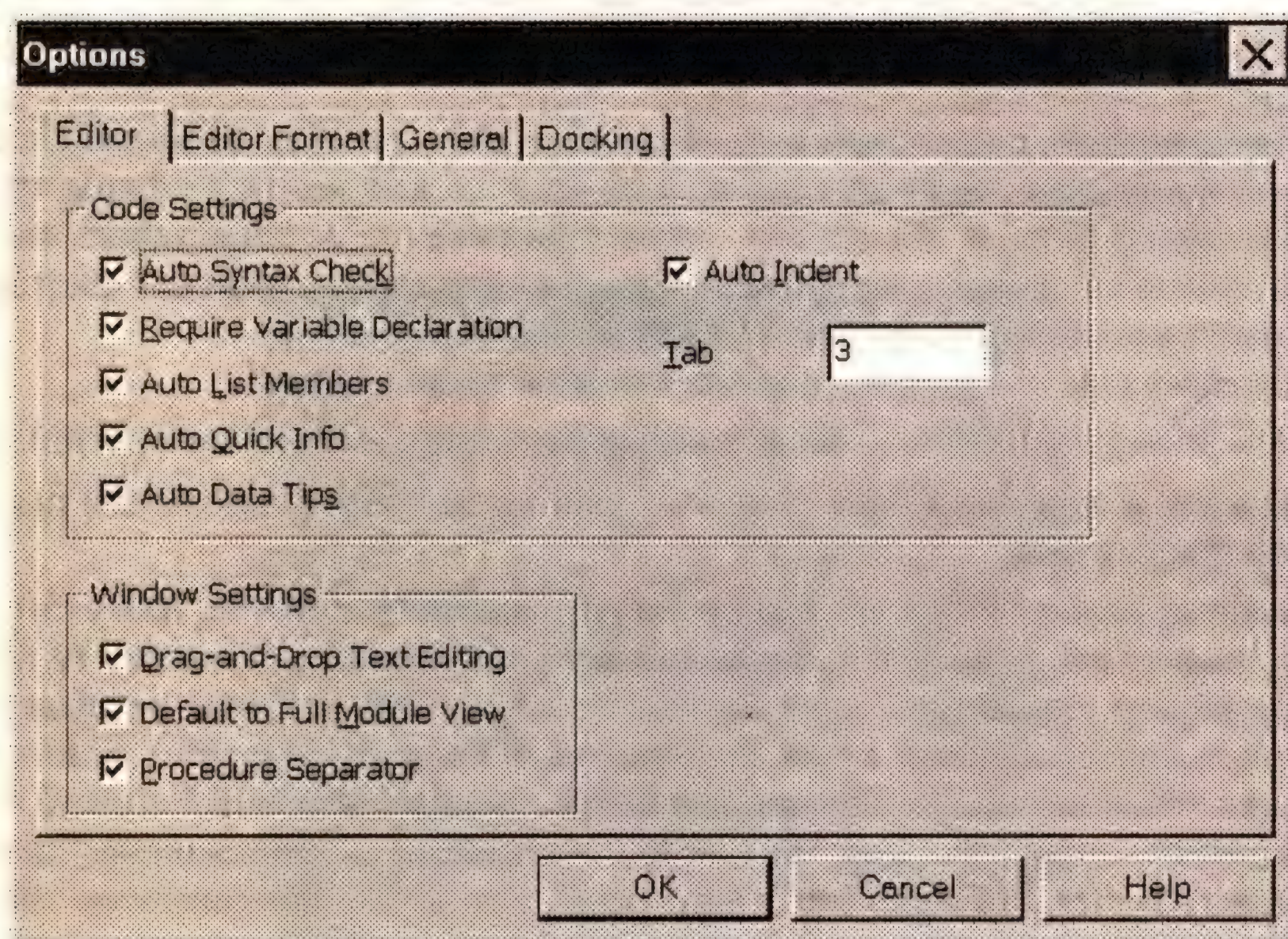


Рис. 4.3. Включение функции автоматической проверки ошибок в диалоговом окне Options

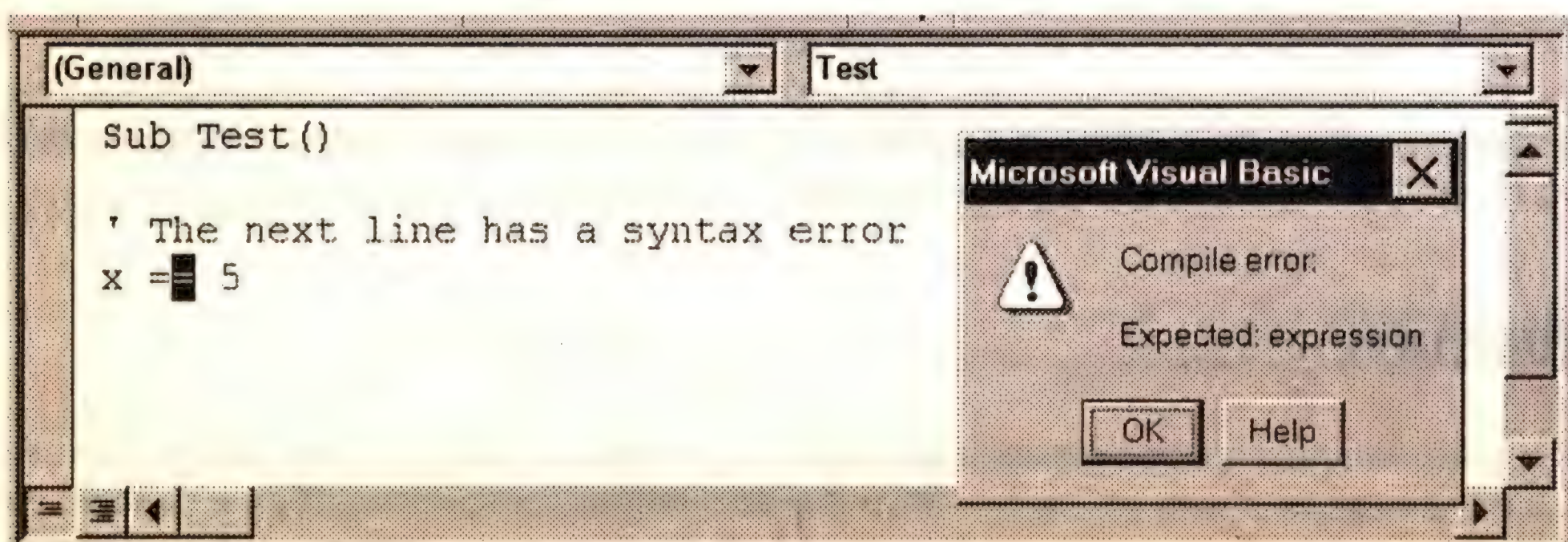


Рис. 4.4. Сообщение о синтаксической ошибке

Обратите внимание на прочие настройки для работы в режиме проектирования. Например, можно указать величину сдвига кода при нажатии клавиши Tab.

На рис. 4.4 показано, что происходит, если, введя ошибочное выражение `x = = 5`, вы попытаетесь перейти на следующую строку. Обратите внимание на то, что компания Microsoft называет такие ошибки *ошибками компиляции (compile error)*. Однако более наглядно называть их просто синтаксическими ошибками.

Ошибки, возникающие во время компиляции

Для начала программу требуется откомпилировать, то есть перевести на язык, понятный компьютеру. Процесс компиляции выполняется автоматически при первом запуске программы или с помощью команды **Compile Project** (Компилировать проект) меню **Debug**.

Если в процессе компиляции кода приложение Excel обнаруживает ошибку, появляется сообщение об этом. Например, код, показанный на рис. 4.5, содержит ошибку компиляции.

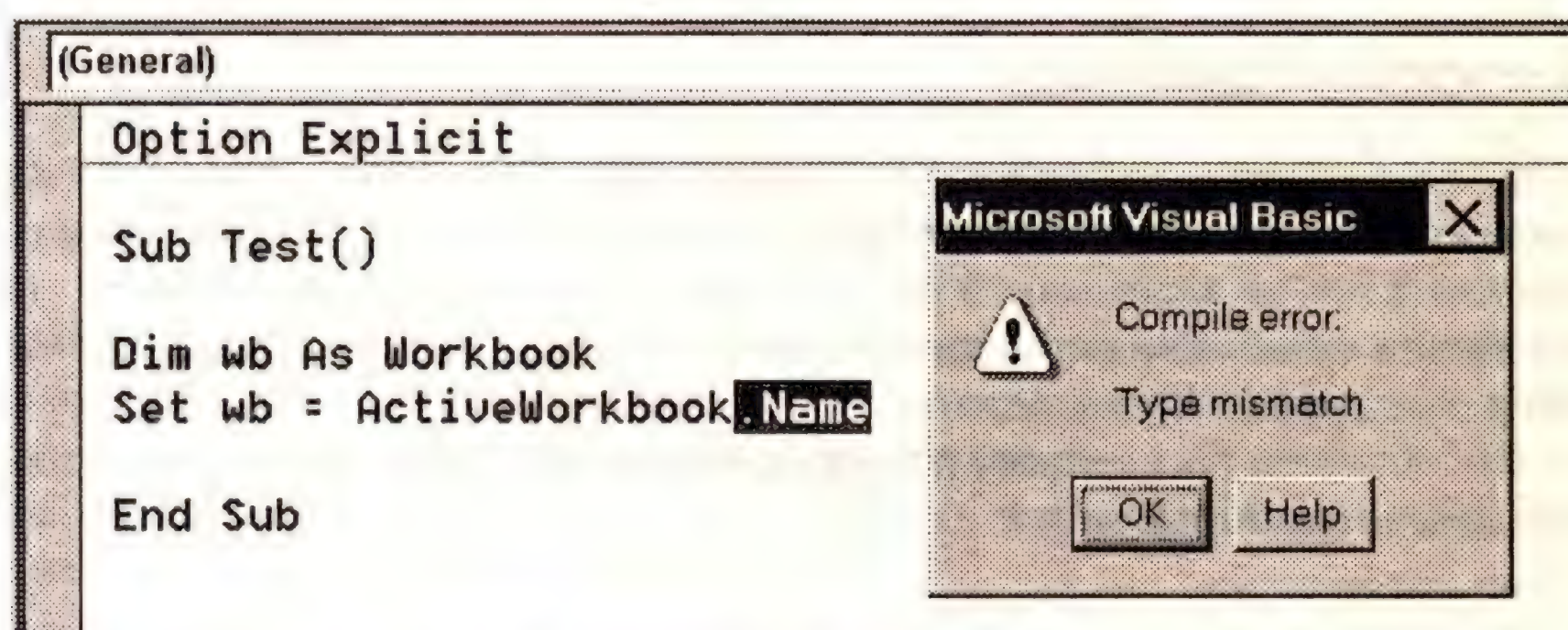


Рис. 4.5. Сообщение об ошибке компиляции

Дело в том, что первая строка

```
Dim wb as Workbook
```

указывает на соответствие переменной типа `Workbook` книге Excel. Однако вторая строка

```
Set wb = ActiveWorkbook.Name
```

присваивает этой переменной не саму активную книгу, а ее имя. Эту ошибку невозможно обнаружить во время проектирования, так как она не является синтаксической. И только в процессе компиляции, когда Excel начинает рассматривать один оператор в контексте другого, ошибка становится очевидной.

Ошибки, возникающие во время запуска программы

Некоторые ошибки можно обнаружить только во время запуска программы. Пример такой ошибки и сообщения о ней показаны на рис. 4.6. В этом примере код `Workbook.Open "d:\temp\ExistNot.xls"`

является попыткой открыть несуществующую книгу. Сообщение об ошибке в этом случае не только информирует, что именно случилось, но и предлагает вариант решения проблемы.

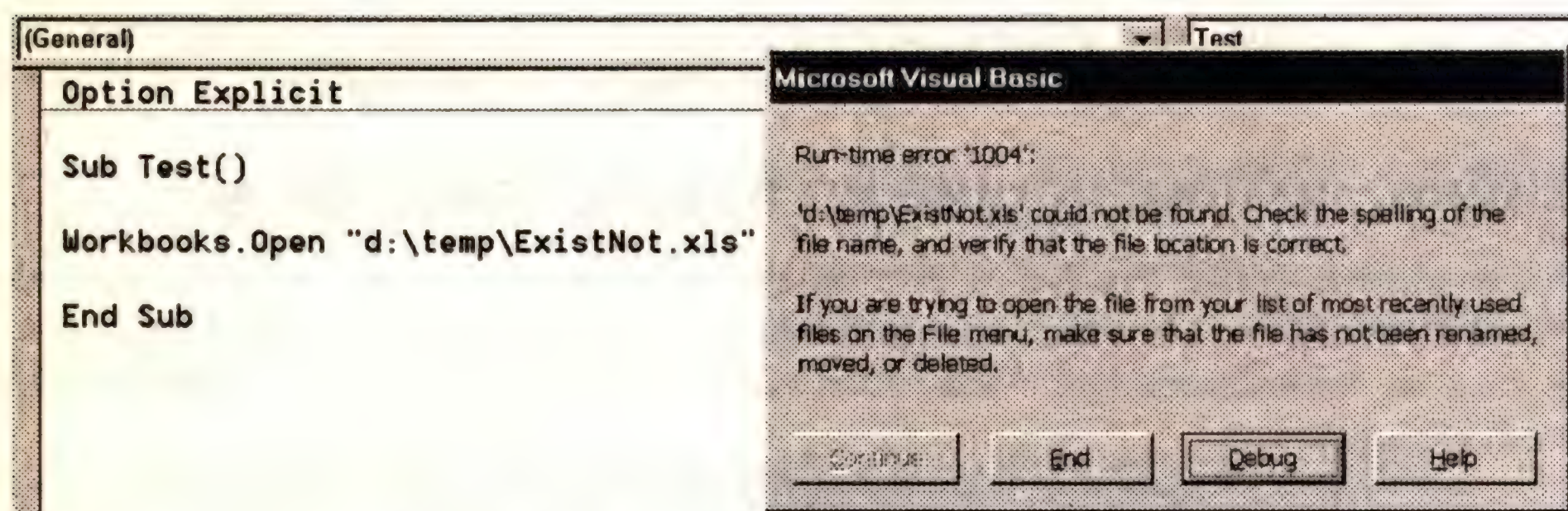


Рис. 4.6. Сообщение об ошибке исполнения

Логические ошибки

Существует еще один тип ошибок, о котором вы должны знать. Логические ошибки можно определить как появление неожиданного и неверного результата. Но так как приложение Excel не знает, какой результат является верным, логические ошибки нельзя считать ошибками периода исполнения в традиционном смысле, хотя проявляются они именно в этот период.

Для наглядности рассмотрим код, предназначенный для вычисления среднего арифметического набора чисел:

```
Dim x(3) As Integer
Dim Ave As Single
x(0) = 1
x(1) = 3
x(2) = 8
x(3) = 5
Ave = (x(0) + x(1) + x(2) + x(3)) / 3
MsgBox "Average is: " & Ave
```

В результате появится окно с сообщением, показанное на рис. 4.7.

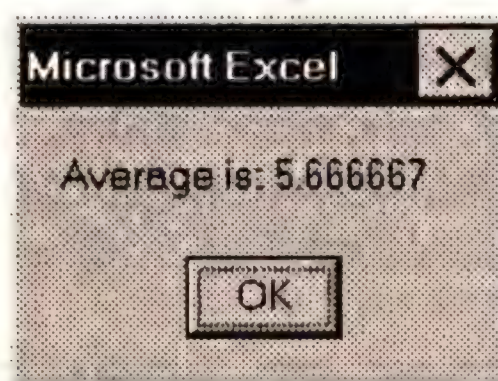


Рис. 4.7. Результат логической ошибки

К сожалению, это неверный результат. Ведь предпоследняя строка программы должна выглядеть так:

```
Ave = (x(0) + x(1) + x(2) + x(3)) / 4
```


Делить сумму нужно на четыре, так как именно такое количество чисел используется для вычисления среднего арифметического. Разумеется, об этой ошибке приложение не сможет вам сообщить, так как «не знает», на какое именно число требовалось разделить сумму.

Как видите, ошибки данного типа являются наиболее опасными, потому что приложение не предупреждает о них, и вы думаете, что все в порядке.

Отладка

Практически любой код содержит ошибки. Ошибки, возникающие в процессе написания кода и компиляции, достаточно легко исправить, так как о них сообщает приложение Excel, выделяя при этом сомнительный код. Логические ошибки распознать и исправить намного сложнее. Именно тут на помощь приходит процесс отладки (*debugging*). Интегрированная среда разработки Excel дает ряд способов поиска ошибок.

Процесс отладки достаточно запутан, и по этому поводу можно написать целую главу. Существуют даже специальные приложения, помогающие в выполнении данной задачи. Однако в большинстве случаев вполне достаточно набора простых техник. В частности, Excel позволяет осуществлять пошаговый просмотр программ, выполняя по одной строке за раз и наблюдая за получаемым эффектом.

Рассмотрим небольшой пример. По возможности, расположите окна на экране, как показано на рис. 4.8. Это упростит процесс отладки, так как вам не потребуется переключаться с окна Excel к редактору VBA и обратно.

Имейте в виду, что строки, начинающиеся с апострофа, являются комментариями и игнорируются приложением Excel.

Листинг 4.1. Иллюстрация процесса отладки

```
Sub Test()  
  
Dim ws As Worksheet  
  
Set ws = ActiveSheet  
  
' Введите значение в ячейку A1  
ws.Cells(1, 1).Value = "sample"  
' Выделите это значение полужирным шрифтом  
ws.Cells(1, 1).Font.Bold = True  
' Скопируйте ячейку в буфер  
ws.Cells(1, 1).Copy  
' Вставьте только значение  
ws.Cells(2, 1).PasteSpecial Paste:=xlValues  
  
End Sub
```

Убедитесь в том, что в данный момент активен чистый лист Excel. Переключитесь в редактор VBA и щелкните в произвольном месте кода. Нажмите клавишу F8, чтобы начать процесс пошагового выполнения программы. В качестве альтернативного варианта можно выбрать команду Step Into в меню Debug.

Нажимайте клавишу F8, останавливаясь на каждой строке и наблюдая, что происходит в окне Excel. Для перехода от окна Excel к редактору VBA и обратно используется комбинация клавиш Alt + F11. По мере пошагового выполнения кода вы увидите, как в ячейке A1 появится слово "sample", затем оно будет выделено жирным шрифтом, скопировано в буфер и вставлено в виде обычного текста в ячейку A2. С этой несложной программы начнется ваше обучение программированию в Excel.

Рассмотрим более подробно инструменты, предназначенные для отладки кода.

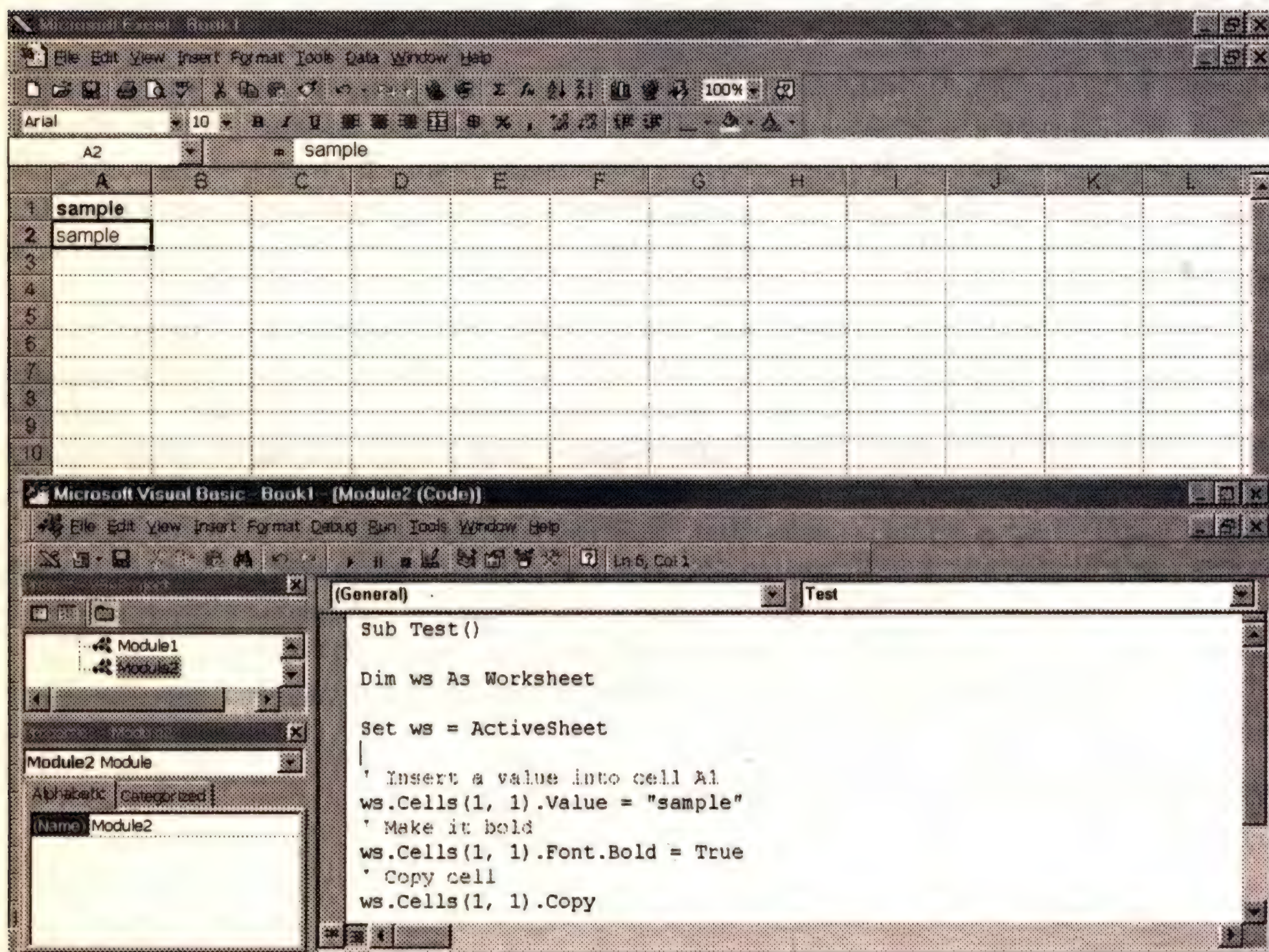


Рис. 4.8. Расположение окон, облегчающее процесс отладки

Пошаговое выполнение программы

Существуют три способа пошагового выполнения программы: *шаг с заходом* (Step Into), *шаг с обходом* (Step Over) и *шаг с выходом* (Step Out). Разница между ними состоит в порядке выполнения инструкций.

Проиллюстрируем все три способа на примере. Первая линия кода процедуры A задает значение ячейки A1 активного листа. Вторая строчка вызывает процедуру B, а третья приводит к выделению содержимого ячейки жирным шрифтом. Процедура B просто меняет размер и шрифт букв в ячейке A1. На данном этапе вас не должен беспокоить синтаксис кода. Достаточно обратить внимание на то, что вторая строчка кода процедуры A вызывает процедуру B.

Листинг 4.2. Различные способы пошагового выполнения программы

```
Sub PrcedureA()  
    ActiveSheet.Cells(1, 1).Value = "Пример"  
    Call ProcedureB  
    ActiveSheet.Cells(1, 1).Font.Bold = True  
End Sub  
  
Sub ProcedureB()  
    ActiveSheet.Cells(1, 1).Font.Size = 24  
    ActiveSheet.Cells(1, 1).Font.Name = "Arial"  
End Sub
```

Шаг с заходом

Шаг с заходом представляет собой выполнение одного оператора (или инструкции) за один раз. Если шаг приводит к вызову еще одной процедуры, вас просто перебросит на ее первую строчку. Например, в вышеприведенном коде шаг с заходом на строку

```
Call ProcedureB
```

переводит управление на строку

```
ActiveSheet.Cells(1, 1).Font.Size = 24
```

Далее начинается пошаговое выполнение процедуры В. После его завершения вы вернетесь на строчку процедуры А, следующую за вызовом процедуры В, то есть:

```
ActiveSheet.Cells(1, 1).Font.Bold = True
```

Шаг с заходом имеет еще одну полезную функцию. Если перейти к пошаговому выполнению программы данным способом в режиме проектирования, процесс начнется со входа в режим прерывания. Это правильный способ начала пошагового выполнения.

Шаг с обходом

Нажатие комбинации клавиш Shift + F8 или выбор команды Step Over в меню Debug также приводит к пошаговому выполнению программы, но в данном случае невозможно осуществить пошаговый просмотр процедур. То есть, дойдя до строки

```
Call ProcedureB
```

приложение моментально выполнит процедуру и перейдет к строке

```
ActiveSheet.Cells(1, 1).Font.Bold = True
```

в процедуре А. Это полезно, если вы уверены, что процедура В не содержит ошибок, и не хотите проверять ее построчно.

Шаг с выходом

Комбинация клавиш Ctrl + Shift + F8 или команда Step Out в меню Debug применяется, когда вы находитесь внутри процедуры и требуется быстро ее завершить. Это полезно, если в процессе пошагового выполнения вызванной процедуры вы решили, что продолжать настоящий процесс дальше не стоит, а имеет смысл вернуться к вызывающей процедуре. Например, если пошаговый просмотр вложенной процедуры был начат по ошибке, достаточно выполнить шаг с выходом, чтобы вернуться в основную программу.

Переход к курсору

В режиме прерывания иногда требуется выполнить несколько строк кода сразу. Именно для этого используется функция перехода к курсору, активируемая комбинацией клавиш **Ctrl + F8** или командой **Run to Cursor** меню **Debug**. Достаточно поместить курсор на строку, следующую за последней строкой фрагмента кода, который вы хотите выполнить, и воспользоваться функцией перехода к курсору.

Переход к следующему оператору

В режиме прерывания можно изменить порядок выполнения программы, поместив курсор на строку, которую требуется выполнить следующей, и нажав комбинацию клавиш **Ctrl + F9** или выбрав в меню **Debug** команду **Set Next Statement**. Это изменит порядок выполнения операторов, когда вы продолжите процесс пошагового выполнения.

Выход из режима отладки

После завершения пошагового выполнения программы у вас есть два выхода. Для возвращения в режим конструирования нужно выбрать команду **Reset** в меню **Run**. Чтобы заставить Excel закончить выполнение текущей программы, нажмите клавишу **F5** или выберите команду **Run** в меню **Run**.

Просмотр выражений

В процессе пошагового выполнения программы иногда требуется узнать значение определенного выражения. В Excel это можно сделать различными способами.

Контрольное значение

Для быстрого просмотра переменной или выражения в режиме прерывания достаточно поместить курсор на переменную и нажать комбинацию клавиш **Shift + F9** или выбрать команду **Quick Watch** в меню **Debug**. На рис. 4.9 показано диалоговое окно **Quick Watch** после выделения выражения $x + 2$. Согласно этому рисунку, на момент вызова функции выражение имело значение 8. Имейте в виду, что если поместить курсор рядом с переменной x , функция **Quick Watch** выдаст значение данной переменной.

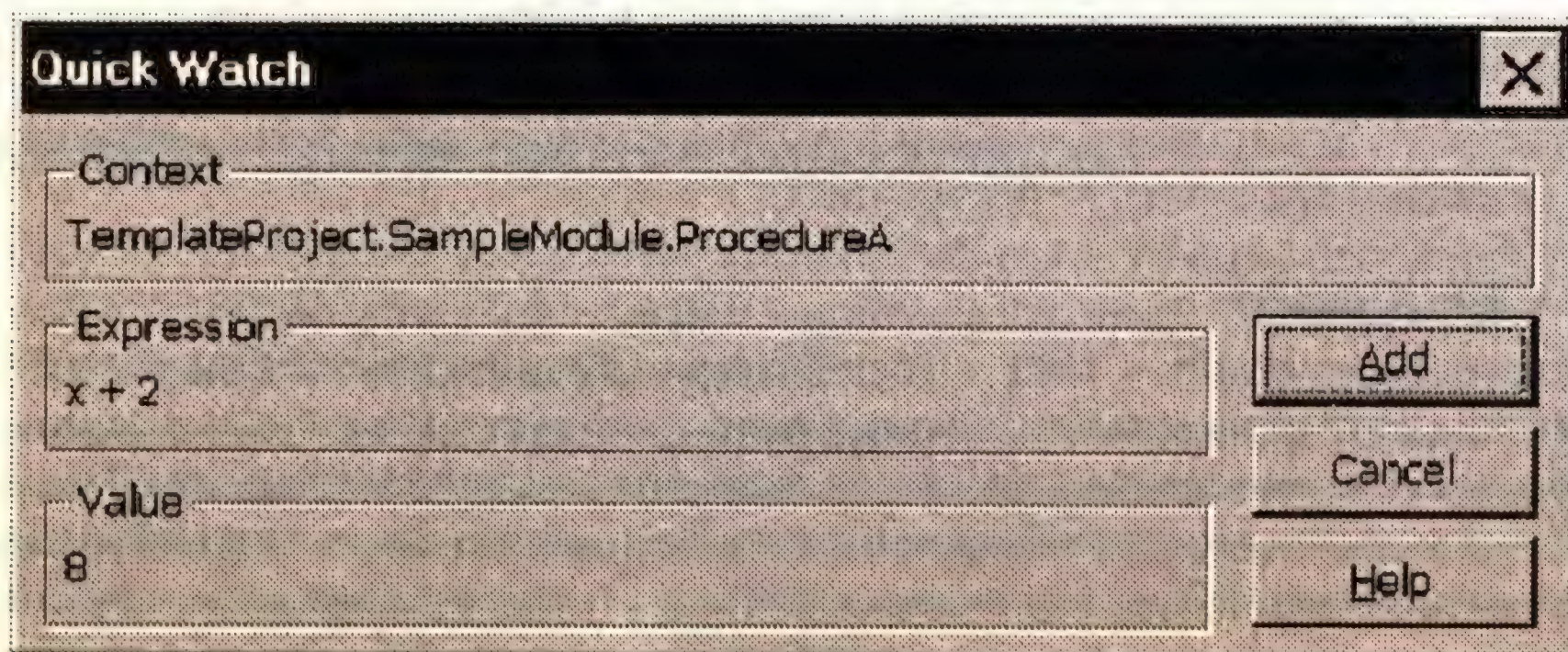


Рис. 4.9. Диалоговое окно Quick Watch

Другим способом быстро определить значение выражения или переменной является установка флажка Auto Data Tip на вкладке Editor диалогового окна Options. В результате, если поместить указатель мыши над переменной или выделенным выражением, через некоторое время появится небольшое желтое окно со значением переменной или выражения.

Окна Locals и Watches

Для просмотра значений переменных используются два окна: Watches и Locals. Они показаны на рис. 4.10.

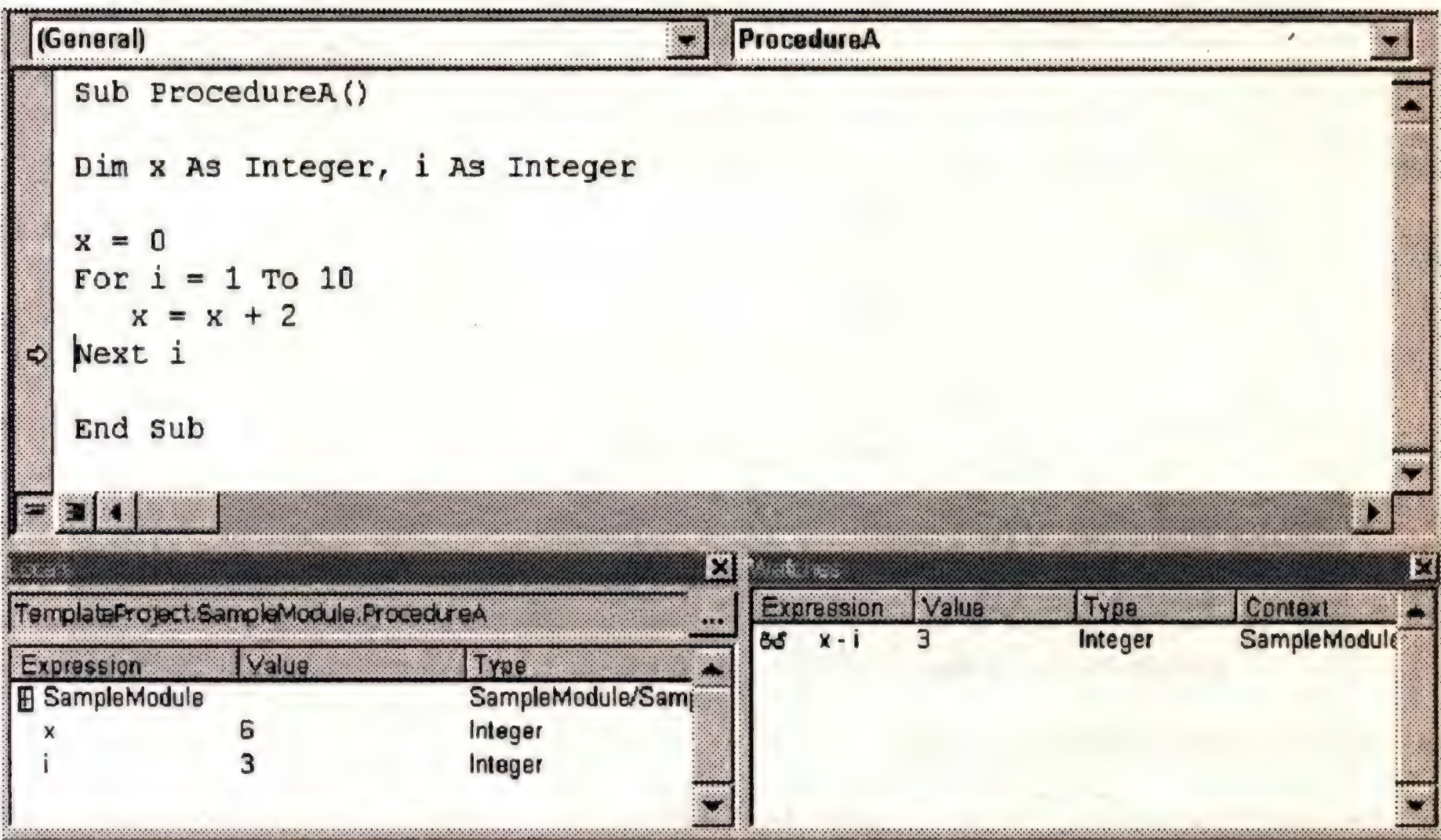


Рис. 4.10. Окна Locals и Watches

Окно Locals показывает значение всех локальных переменных. Это переменные, определяемые внутри текущей процедуры и, следовательно, неприменимые к другим процедурам. О них мы подробно поговорим в следующей главе.

Окно Watches отображает значения всех переменных или выражений, которые вы в него поместите. Приложение Excel автоматически обновляет выражения в этом окне после выполнения каждой строки кода и действует согласно указанному типу контрольных значений.

Чтобы поместить переменную или выражение в окно Watches, используется команда Add Watch меню Debug. Она приводит к появлению диалогового окна, показанного на рис. 4.11.

Вы можете ввести переменную или выражение в поле Expression. Переключатель Watch Type устанавливается в одно из трех положений:

- Watch Expression. Выражение помещается в окно Watches, что дает возможность следить за его изменением по мере выполнения кода. Например, для выражения $x > 6$ в окне Watches существуют только два значения — True или False — в зависимости от величины параметра x .
- Break When Value is True. Если переменная приобретает значение True, приложение переходит в режим прерывания и останавливает выполнение кода. Для

выражения $x > 6$ эта ситуация наступает, когда значение переменной x становится больше 6.

- Break When Value Changes. Приложение переходит в режим прерывания при любом изменении контрольного значения. В рассматриваемом случае происходит переход от True к False и наоборот.

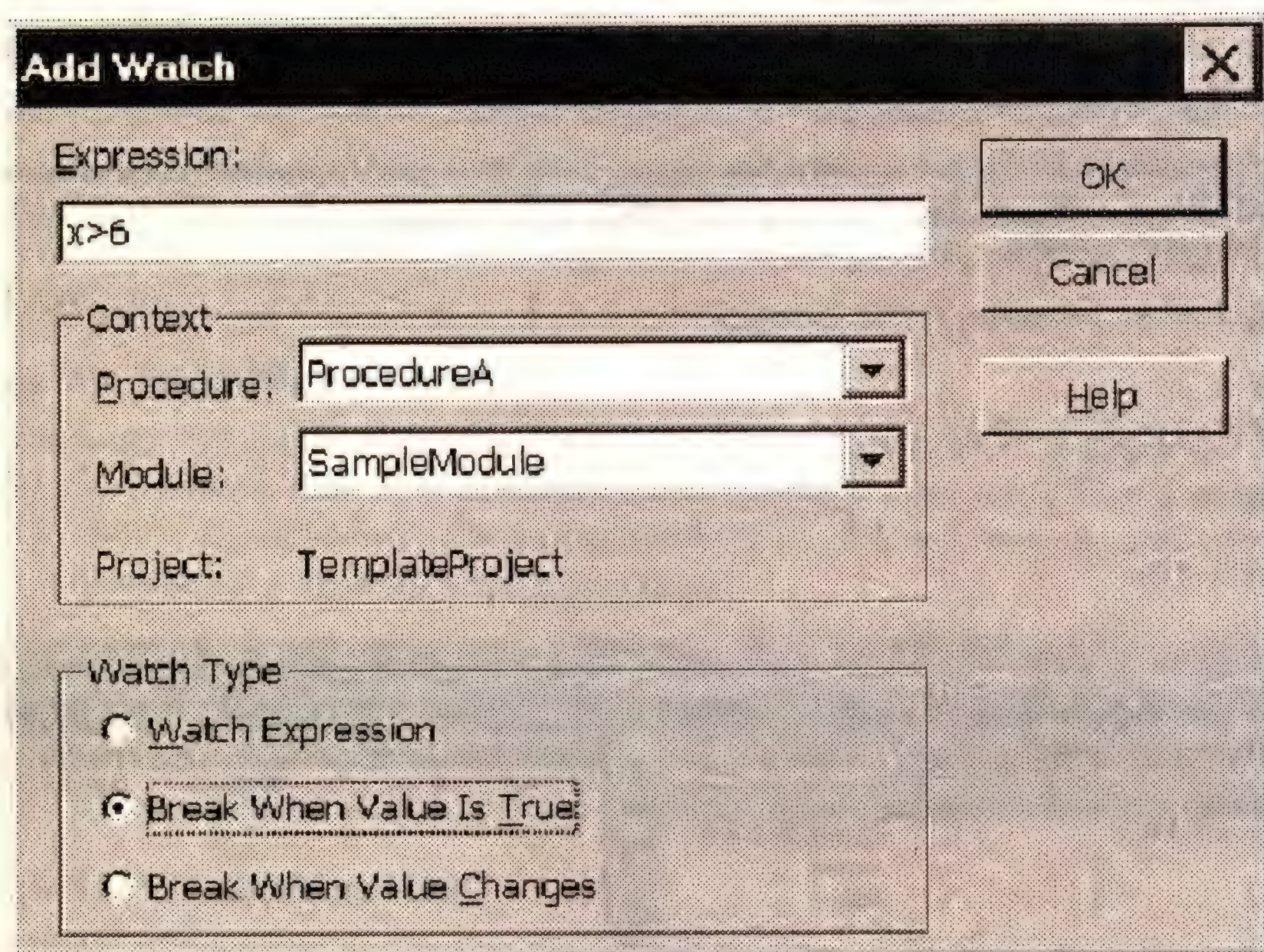


Рис. 4.11. Диалоговое окно Add Watch

Вместе различные режимы пошагового просмотра и типы контрольных значений дают мощный механизм отладки программ, с которым вам предстоит сталкиваться достаточно часто.

Макросы

Раньше макросы представляли собой несколько строк кода, которым соответствовали клавиатурные комбинации. Нажатие нужной комбинации приводило к выполнению кода.

В наши дни макросы являются более сложными объектами. Теперь это своего рода процедура, не имеющая параметров. Подробную информацию о параметрах и процедурах вы найдете в главе 6.

Запись макросов

Приложение Excel дает возможность быстрой записи очень простых макросов. Для этого нужно выбрать команду Tools ► Macro ► Record New Macro (Сервис ► Макрос ►

Начать запись) в главном меню Excel и ввести в появившееся диалоговое окно код, который будет преобразован в процедуру на языке VBA.

Например, представим макрос, который находит слово “Макрос” и меняет его на слово “Подпрограмма”. Если посмотреть окно Project (Проект) для проекта, в котором был создан макрос, обнаружится новая процедура в стандартном модуле:

```
Sub Macro1()  
.  
· Макрос Macro1  
· Макрос записан 9/13/98 Стивеном Романом  
.  
.  
Cells.Replace What:="Макрос". Replacement:="Подпрограмма". _  
    LookAt:=xlPart, SearchOrder:=xlByRows, MatchCase:=False  
End Sub
```

Именно такой код потребовалось бы написать для выполнения операции поиска и замены.

В определенных ситуациях данный способ записи макросов является очень полезным обучающим инструментом. Если вы не можете понять, какой код соответствует тому или иному действию, достаточно записать соответствующий макрос и скопировать получившийся код.

Однако не стоит приходить в преждевременный восторг по поводу такого способа программирования. Ведь это не панацея от всех проблем. В частности, записанный с помощью макрорегистратора код страдает отсутствием универсальности. Например, он может быть связан с переменными, выделенными в момент записи макроса, но его нельзя использовать в общем случае.

Кроме того, таким способом можно записывать только очень простые процедуры, в то время как наиболее полезные программы Excel являются достаточно сложными.

Ну и наконец, автоматически созданный код изобилует лишними деталями, что замедляет процесс его выполнения.

Запуск макросов

Для запуска макроса с помощью пользовательского интерфейса Excel используется команда Tools ► Macro ► Macros (Сервис ► Макрос ► Макросы) или комбинация клавиш Alt + F8. Появляется диалоговое окно, показанное на рис. 4.12, со списком всех макросов в текущей книге или во всех открытых книгах.

Здесь можно запускать, редактировать, создавать и удалять макросы. Щелчок на кнопках Edit (Изменить) и Create (Создать) приводит к появлению окна редактора Visual Basic.

В диалоговом окне Macro (Макрос) отображаются все написанные вами макросы. Однако есть несколько нюансов. В раскрывающемся списке появляются только имена уникальных макросов. Если же имя не является уникальным, к нему присоединяется имя модуля, к которому этот макрос принадлежит. Например:

```
Sheet5.ScrollChartTypes
```

на рис. 4.12. Обратите внимание на наличие еще одного макроса с именем ScrollChartTypes.

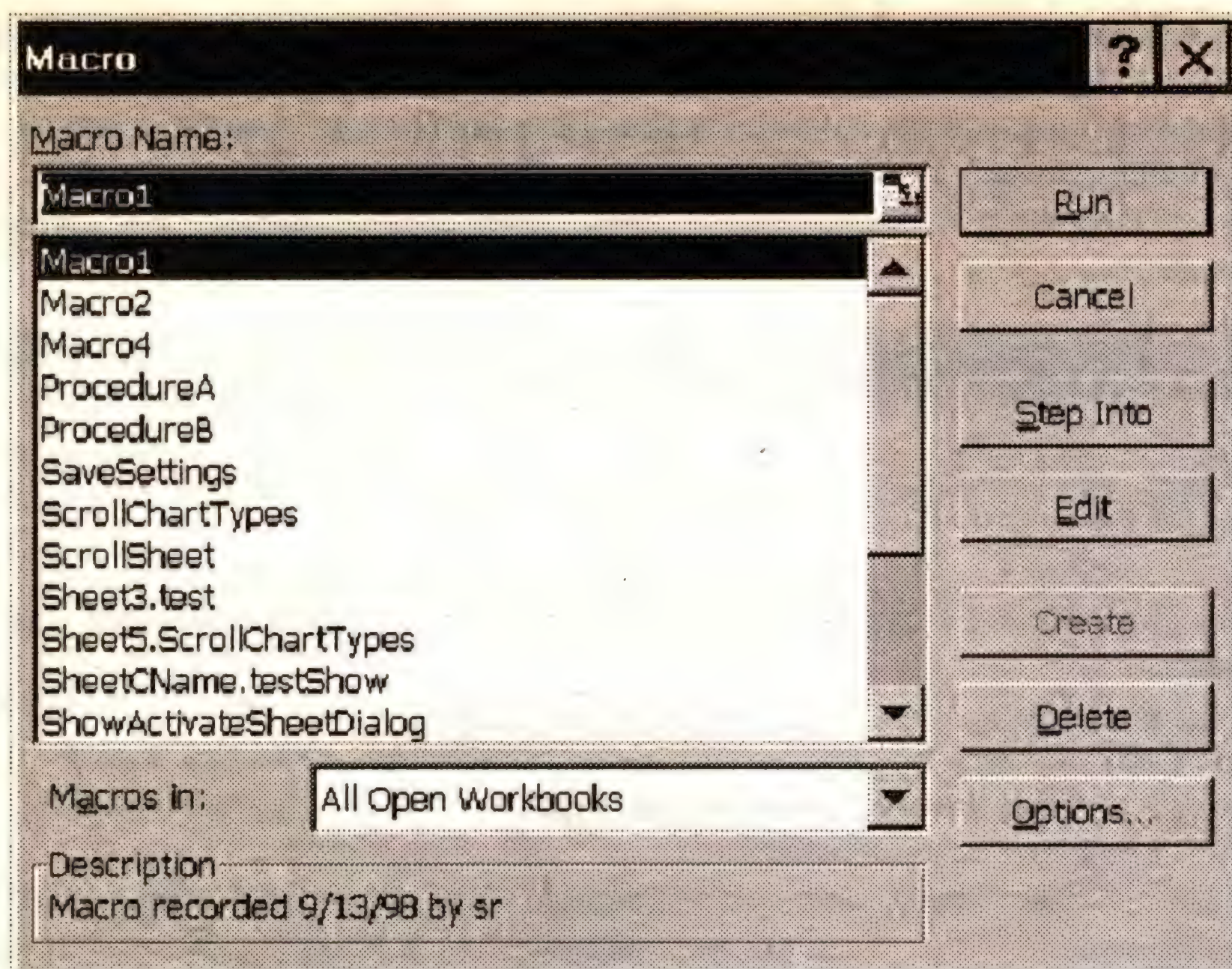


Рис. 4.12. Диалоговое окно Macro

Если вы не хотите, чтобы имя макроса появлялось в списке диалогового окна Macro (Макрос), его требуется объявить с ключевым словом `Private`:

```
Private Sub HideThisMacro()
```

Подробно процедуры уровня модуля и уровня проекта будут обсуждаться в главе 6.

За годы работы я создал множество макросов. По собственному опыту могу сказать, что со временем имена забываются, что приводит к проблемам в поисках определенного макроса. Я посоветовал бы вам выработать систему именования. Я начинаю все имена с названия категории. Например, все макросы для работы с листами Excel начинаются с буквосочетания `Wks`:

```
Wks_Sort  
Wks_Compare  
Wks_Print
```


Часть II. Язык программирования VBA

- Глава 5. Переменные, типы данных и константы
- Глава 6. Функции и подпрограммы
- Глава 7. Встроенные функции и операторы
- Глава 8. Управляющие операторы

5

Переменные, типы данных и константы

В главах этой части мы поговорим об основах языка VBA, необходимых для программирования в среде различных приложений Microsoft Office. В процессе обсуждения будет рассмотрено множество примеров кода. Надеюсь, вы не пожалеете времени и проверите их действие на практике.

Комментарии

Мы уже говорили о том, насколько важны комментарии. Любой текст, следующий за апострофом, представляет собой комментарий, и, соответственно, игнорируется приложением. Например, в нижеприведенном фрагменте кода комментариями являются первая строка и часть третьей:

```
'Объявление переменной типа String  
Dim WksName as String  
WksName = ActiveSheet.Name 'Укажите имя активного листа
```

В процессе отладки кода имеет смысл превращать в комментарии некоторые строки, отменяя их выполнение. Потом их можно снова сделать активными, удалив апострофы. Кнопки **Comment Block** и **Uncomment Block** панели инструментов **Edit** позволяют легко сделать строку или группы выделенных строк комментарием и удалить у них символ комментария соответственно. К сожалению, для этих команд не существует заданных по умолчанию клавиатурных комбинаций, но их можно добавить самостоятельно.

Продолжение строки

Синтаксис языка VBA таков, что иногда приходится писать длинные, трудночитаемые строки кода, для просмотра которых необходима горизонтальная прокрутка. Для таких случаев компания Microsoft недавно разработала символ продолжения строки. Это знак подчеркивания, которому обязательно должен предшествовать пробел и после которого не должно быть никаких символов, в том числе и комментариев. Например, следующий код

```
ActiveSheet.Range("A1").Font.Bold = _  
True
```


интерпретируется приложением как одна строка. Имейте в виду, что символ продолжения строки не может быть вставлен внутрь строковой константы, заключенной в кавычки.

Константы

В языке VBA используются два типа констант. *Литерная константа (literal constant)* представляет собой определенное значение, например число, дату или текстовую строку, вид которого не меняется. Строковые константы заключаются в двойные кавычки, например «Donna Smith», а даты записываются между знаками числа, например #1/1/96#.

Назначением следующего кода является сохранение даты в виде переменной dt:

```
Dim dt As Date  
dt = #1/2/97#
```

Для описания *символьной константы (symbolic constant)* в программе используется ключевое слово Const. Например:

```
Const InvoicePath = "d:\Invoices\"
```

В результате приложение заменит любой экземпляр слова InvoicePath строкой d:\Invoices\. Соответственно, InvoicePath — это константа, так как ее значение не меняется. Но так как она используется не в том виде, как была записана изначально, к литерным константам ее отнести нельзя.

Если позднее вы решите поменять значение d:\Invoices\ на d:\OldInvoices\, достаточно будет переопределить константу InvoicePath.

Хорошим тоном в программировании считается определение символьных констант в начале процедуры, в которой они будут использоваться, или в разделе Declarations (Описание) модуля кода. Это делает код более читаемым и облегчает контроль над его выполнением.

Кроме символьных констант, определяемых с помощью оператора Const, в языке VBA существует около 700 встроенных констант, имена которых начинаются с буквосочетания vb. Язык VBA для Excel добавляет еще 1266 символьных констант, имена которых начинаются с буквосочетания xl. Со многими из них вы столкнетесь в процессе чтения данной книги.

Среди часто используемых констант VBA следует отметить vbCrLf, соответствующую переносу на другую строку и возврату каретки, и vbTab, действие которой эквивалентно нажатию клавиши Tab.

Перечисления

Компания Microsoft разработала структуру, предназначенную для разбиения множества констант на категории. Эта структура называется *перечислением (enum)*. Список перечислений можно получить в окне Object Browser. Например, одно из 152 перечислений Excel используется в методе автозаполнения. Вот как оно выглядит:

```
Enum XLAutoFillType  
    xlFillDefault = 0  
    xlFillCopy = 1
```



```

xlFillSeries = 2
xlFillFormats = 3
xlFillValues = 4
xlFillDays = 5
xlFillWeekdays = 6
xlFillMonths = 7
xlFillYears = 8
xlLinearTrend = 9
xlGrowthTrend = 10
End Enum

```

В документации по Excel это перечисление неправильно называют `XlFillType`. Обратите внимание на то, что имена перечислений начинаются с буквосочетания `Xl`.

Соответственно, следующий код приведет к заполнению первых семи ячеек первого ряда активного листа названиями дней недели. При этом предполагается, что в первой ячейке будет слово `Monday` (понедельник).

```
ActiveSheet.Range("A1").AutoFill ActiveSheet.Range("A1:G1").xlFillDays
```

Этот код можно записать и немного по-другому:

```
ActiveSheet.Range("A1").AutoFill ActiveSheet.Range("A1:G1"), 5
```

но он не настолько нагляден, как в первом случае. Так как перечисления встроены в Excel, их не требуется каждый раз описывать в программах. В принципе, существует возможность создать собственные перечисления, но в подавляющем большинстве случаев это не нужно, так как встроенных версий вполне достаточно.

В качестве еще одного примера рассмотрим встроенное перечисление для постоянного значения, которое возвращается, когда пользователь закрывает окно сообщения.

```

Enum VbMsgBoxResult
    vbOK = 1
    vbCancel = 2
    vbAbort = 3
    vbRetry = 4
    vbIgnore = 5
    vbYes = 6
    vbNo = 7
End Enum

```

В результате, когда пользователь нажимает кнопку `OK` в диалоговом окне (предполагается, что таковая присутствует), приложение возвращает значение `vbOK`. О том, как использовать это значение, мы поговорим чуть позже.

В языке VBA также определены символьные константы, задающие тип кнопок в диалоговых окнах. Они содержатся в следующем перечислении:

```

Enum VbMsgBoxStyle
    vbOKOnly = 0
    vbOKCancel = 1
    vbAbortRetryIgnore = 2
    vbYesNoCancel = 3
    vbYesNo = 4
    vbRetryCancel = 5
End Enum

```

Впрочем, в данном перечислении есть дополнительные константы, не показанные в настоящий момент. Для иллюстрации его работы рассмотрим следующий код:


```
If MsgBox("Proceed?", vbOKCancel) = vbOK Then  
    ' здесь находится код, который выполняется при нажатии кнопки OK  
Else  
    ' здесь находится код, который выполняется при нажатии любой другой кнопки  
End If
```

Первая строка этого кода приводит к появлению диалогового окна с вопросом **Proceed?** (Продолжить) и кнопками **OK** и **Cancel**, как показано на рис. 5.1.

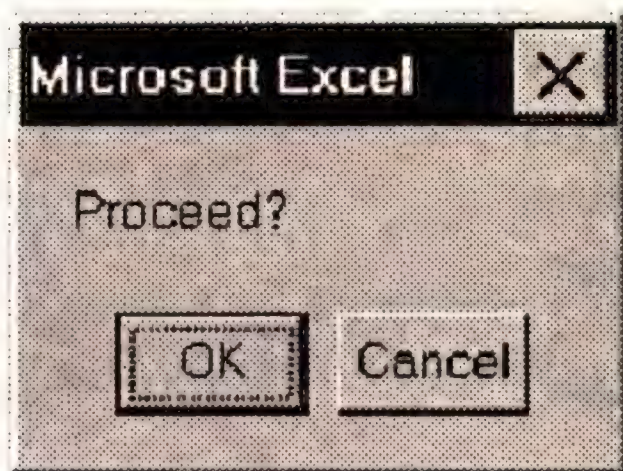


Рис. 5.1. Пример диалогового окна, содержащего сообщение

Если пользователь нажмет кнопку **OK**, приложение вернет константу `vbOK`; в противном случае возвращена будет константа `vbCancel`. Таким образом, оператор `If`, который вы видите в первой строке, дает возможность выбора между двумя вариантами. Но о нем мы более подробно поговорим в главе 8.

На случай, если вы еще не убедились в ценности символьных констант, приведу пример перечисления для различных цветов:

```
Enum ColorConstants  
    vbBlack = 0  
    vbBlue = 16711680  
    vbMagenta = 16711935  
    vbCyan = 16776960  
    vbWhite = 16777215  
    vbRed = 255  
    vbGreen = 65280  
    vbYellow = 65535  
End Enum
```

Только представьте себе, что вместо кода

```
ATextBox.ForeColor = vbBlue
```

вам придется вводить код

```
ATextBox.ForeColor = 16711680
```

Вам требуются дополнительные примеры?

Переменные и типы данных

Переменной называется место в памяти, которое может хранить значение определенного типа. Это значение может изменяться по мере выполнения программы, именно поэтому в данном случае и используется слово «переменная».

В языке VBA каждая переменная относится к определенному типу данных. Например, переменная типа `String` может содержать текстовые строки и называется *строковой переменной* (*string variable*). Переменной типа `Integer` можно присва-

ивать целые числа. Соответственно, она называется *целочисленной переменной* (*integer variable*). Список типов данных языка VBA приведен в табл. 5.1

Таблица 5.1. Типы данных в VBA

Тип данных	Размер в памяти	Диапазон значений
Byte	1 байт	От 0 до 255
Boolean	2 байта	True или False
Integer	2 байта	От -32 768 до 32 767
Long (long integer)	4 байта	От -2 147 483 648 до 2 147 483 647
Single (single-precision real)	4 байта	Приблизительно от -3.4E38 до 3.4E38
Double (double precision real)	8 байт	Приблизительно от -1.8E308 до 4.9E324
Currency (scaled integer)	8 байт	Приблизительно от -922 337 203 685 477 5808 до 922 337 203 685 477 5807
Date	8 байт	От 1/1/100 до 12/31/9999
Object	4 байта	Любая ссылка на объект
String	Длина переменной: 10 байт + длина строки Фиксированная длина: длина строки	Длина переменной около 2 миллиардов Фиксированная длина: до 65 400
Variant	16 байт для чисел 22 байта + длина строки	Численные, как у Double Строковые, как у String
User-Defined	Разные	

Описание переменных

Под описанием переменной подразумевается указание типа данных, к которому она принадлежит. Для этого используется ключевое слово Dim. Вот ряд примеров:

```
Dim Name As String
Dim Holyday As Date
Dim Age As Integer
Dim Height As Single
Dim Money As Currency
Dim wbk As Workbook
Dim ch As Chart
```

Общий вид описания переменной выглядит следующим образом:

```
Dim Имя переменной As Тип данных
```

Если пропустить описание переменной или не указать, к какому типу данных она принадлежит, приложение будет считать ее переменной типа Variant. Как видно из табл. 5.1, это не очень рациональная трата пространства, так как эти переменные занимают в памяти больше места, чем переменная любого другого типа.

Например, переменная типа Integer занимает 2 байта, в то время как переменная типа Variant, содержащая то же самое целое число, занимает 16 байт. Сложные

программы насчитывают сотни, а иногда и тысячи переменных, так что излишняя трата памяти может быть весьма значительной. Именно поэтому желательно всегда описывать переменные.

При этом управление переменными типа Variant требует большего количества ресурсов по сравнению с обработкой переменных других типов, что уменьшает производительность.

Для экономии места можно описывать несколько переменных в одной строке. Например, следующий код описывает три переменные:

```
Dim Age As Integer, Name As String, Money As Currency
```

Имейте в виду, что описание вида

```
Dim Age, Height, Weight As Integer
```

вполне имеет право на существование, но переменные Age и Height будут принадлежать к типу Variant, а не Integer. Другими словами, требуется указывать тип каждой переменной в отдельности.

Кроме того, допустимо обозначать тип переменной, добавляя к ее имени специальные символы, перечисленные в табл. 5.2. Лично я предпочитаю не употреблять эти символы, хотя они и помогают экономить место.

Таблица 5.2. Символы для описания типа переменных

Символ	Тип переменной
%	Integer
&	Long
!	Single
#	Double
@	Currency
\$	String

Например, следующий код описывает переменную Name\$ как строковую:

```
Dim Name$
```

Затем можно написать:

```
Name$ = "Donna"
```

Напоследок упомянем, что хотя Excel позволяет описывать переменные и константы в произвольно выбранном месте, хорошим тоном является размещение описаний в начале процедуры.

Зачем нужно подробное описание переменных

Я уже упомянул, что переменные типа Variant приводят к нерациональному расходу пространства и снижению производительности. Но существует еще одна, более важная причина необходимости описания всех переменных. Это связано с опечатками, которые все делают время от времени. И если вы случайно сделаете опечатку в имени переменной, приложение решит, что была создана еще одна переменная.

Чтобы проиллюстрировать опасность подобной ситуации, рассмотрим процедуру NewBook, которая открывает книгу, вносит в нее изменения, спрашивает у пользователя, под каким именем сохранить измененную книгу, и сохраняет ее.

Листинг 5.1. Пример процедуры с опечаткой

```
Sub NewBook()  
  
Dim Wbk As Workbook  
Dim WbkName As String  
  
' Откройте книгу  
Set Wbk = Workbooks(1)  
  
' Укажите имя книги  
WbkName = Wbk.Name  
  
' Здесь находится код, меняющий содержимое книги  
' Пользователя спрашивают о новом имени документа  
WkbName = InputBox ("Введите имя книги " & WbkName)  
  
' Сохранение книги  
Wbk.SaveAs WbkName  
  
End Sub
```

Опечатка находится в строке

```
WkbName = InputBox ("Введите имя книги " & WbkName)
```

Неописанную переменную `WkbName` приложение определит как новую переменную типа `Variant`. Мало того, новое имя будет назначено именно этой переменной, а измененный документ сохранится под старым именем. Соответственно, содержимое исходной книги будет переписано без предупреждения.

Оператор `Option Explicit`

Как избежать описанной выше проблемы? Достаточно сделать так, чтобы приложение не запускало программу, в которой присутствуют неописанные переменные. Для этого нужно поместить оператор

```
Option Explicit
```

в область описаний модуля. Но об этом можно легко забыть. Если вы предпочитаете, чтобы эта инструкция вставлялась автоматически, установите флажок `Require Variable Declaration` на вкладке `Editor` диалогового окна `Options`. Я настоятельно рекомендую вам сделать это!

А теперь перейдем к описанию типов данных, перечисленных в табл. 5.1.

Численные типы данных

К численным типам данных относятся `Integer`, `Long`, `Single`, `Double` и `Currency`. Это целые числа, продленные целые числа, действительные числа одинарной точности, действительные числа двойной точности и денежные единицы.

Логические типы данных

Переменные типа `Boolean` принимают одно из двух значений — `True` или `False`. Это крайне полезный тип данных, недавно появившийся в языке VBA. До этого считалось, что ноль имеет значение `False`, а все ненулевые величины — значение `True`. Примеры этого до сих пор можно найти в старых программах.

Строковые типы данных

Строкой называется последовательность символов. Впрочем, существует пустая строка, не содержащая символов вообще. Строка может состоять как из обычных текстовых символов (буквы, цифры и знаки препинания), так и из специальных управляющих символов, например `vbCrLf` (возврат каретки и переход к началу новой строки). Как уже упоминалось, константы данного типа заключаются в двойные кавычки. Пустая строка также помечается парой кавычек:

```
EmptyString = ""
```

В языке VBA есть два типа строковых переменных — с фиксированной и с изменяемой длиной. Первые определяются следующим образом:

```
Dim Имя переменной As String * Длина строки
```

Например, следующий оператор описывает строку с фиксированной длиной в 10 символов:

```
Dim sName As String * 10
```

Рассмотрим код, содержащий две строки:

```
Dim s As String * 10
```

```
s = "test"
```

```
Debug.Print s & "/"
```

В результате получаем

```
test /
```

Как видите, в строку были добавлены пробелы, чтобы она достигла заданной длины.

Переменные с изменяемой длиной строки описываются следующим образом:

```
Dim Имя переменной As String
```

Например, код

```
Dim s As String
```

```
s = "test"
```

```
Debug.Print s & "/"
```

```
s = "another test"
```

```
Debug.Print s & "/"
```

даст следующий результат

```
test/
```

```
another test/
```

Переменные с изменяемой длиной строки используются намного чаще переменных с фиксированной длиной строки. Впрочем, последние имеют весьма специфическую область использования, рассказ о которой выходит за рамки данной книги.

Дата и время

Переменные типа `Date` занимают по 8 байт и хранятся в виде десятичных чисел (с плавающей точкой), которые представляют даты с 1 января 100 года до 31 декабря 9999 года (в данном случае отсутствует проблема 2000 года) и время от 0:00:00 до 23:59:59.

Как упоминалось ранее, численные данные заключаются между знаками #, но в случае с датами также можно использовать и строковый формат. Например, все перечисленные ниже переменные определены корректно:

```
Dim dt As Date
dt = #1/2/98#
dt = "January 12,2001"
dt = #1/1/95#
dt = #12:50:00 PM#
dt = #1/13/76 12:50:00 PM#
```

В языке VBA имеется множество функций для управления датами и временем. Информацию о них можно найти в справочном файле VBA, начав с запроса Data Data Type.

Тип данных Variant

Переменные типа Variant допускают присвоение данных любого типа кроме строк с фиксированной длиной и пользовательских типов данных. Вы уже знакомы с преимуществами и недостатками этих переменных и знаете, почему их нежелательно использовать в программе.

Объектные типы данных

В языке VBA существует ряд дополнительных типов данных, попадающих в категорию объектных переменных. Их полный список вы найдете в главе, посвященной объектной модели Excel. Вот список наиболее выдающихся объектов:

Объекты, связанные с диаграммами

Axis	ChartTitle	Legend	Series
AxisTitle	DataLabel	LegendEntry	SeriesCollection
Chart	DataTable	LegendKey	TickLabels
ChartArea	Floor	PlotArea	Walls
ChartColorFormat	GridLines	Point	

Объекты, связанные со сводными таблицами

PivotCache	PivotField	PivotFormula	PivotItem	PivotTable
------------	------------	--------------	-----------	------------

Объекты общего назначения

Comment	Font	Range	Workbook
FillFormat	Outline	Sheets	Worksheet
Filter	PageSetup	Window	WorksheetFunction

Соответственно, переменные можно описывать следующим образом:

```
Dim wb As Workbook
Dim wks As Worksheet
Dim chrt As Chart
Dim ax As axis
Dim pf As PivotField
```


Как вы вскоре убедитесь, большая часть этой книги посвящена объектной модели Excel. И именно эти объекты дают возможность программно управлять данным приложением.

Описание с помощью оператора As Object

Описать объектную переменную можно и с помощью оператора As Object, как показано в нижеприведенном примере:

```
Dim chrt As Object
```

Такой способ описания можно увидеть время от времени, но он не настолько эффективен, как описание с указанием конкретного типа объекта, например:

```
Dim chrt As Chart
```

Если приложение не может сразу определить, к какому типу относится переменная chrt, на это приходится тратить время выполнения программы. Данный процесс называется *динамическим связыванием (late binding)*. Иногда он значительно замедляет выполнение программы. Более подробную информацию на эту тему можно найти в приложении Е. Пока достаточно запомнить, что желательно использовать описание с указанием конкретного типа переменной.

Об объектных переменных мы поговорим в главе 9, а на данном этапе остается только упомянуть об операторе Set, который время от времени встречается в примерах кода.

Оператор Set

Описание объектных переменных ничем не отличается от описания переменных других типов. Вот два варианта описания:

```
Dim int As Integer ' Описание неobjектной переменной
```

```
Dim chrt As Chart ' Описание объектной переменной
```

Однако синтаксис присваивания объектным переменным некоторого значения отличается от синтаксиса присваивания значения переменным других типов. В частности, требуется использовать ключевое слово Set. Следующий пример кода назначает активную в данный момент диаграмму переменной chrt:

```
Set chrt = ActiveChart
```

Если диаграмма не является активным объектом на момент выполнения данной строки кода, переменной chrt присваивается значение Nothing.

Массивы

Массивом (array) называется пронумерованная группа значений одного типа. Их имена различаются только по индексам. Например, массив, предназначенный для сохранения первых ста ячеек первого ряда листа, описывается следующим образом:

```
Dim Cell(1 To 100) As Range
```

В объектной модели Excel не существует объекта Cell. Это специальный объект Range. Переменной массива в данном случае выступает Cell. Нижняя граница индексов массива равна 1, а верхняя — 100. Каждая из следующих переменных является переменной Range:

```
Cell(1). Cell(2). ..., Cell(100)
```


Имейте в виду, что если в описании пропустить первый индекс, например:

```
Dim Cell(100) As Range
```

в качестве первого индекса автоматически будет взято число ноль, и общее количество элементов массива станет равным 101.

Только представьте себе, как сложно было бы описывать сто переменных по отдельности, и вы поймете, какое замечательное преимущество имеют массивы. Вскоре вы познакомитесь со способами обработки элементов массива. Например, следующий код приводит к выделению жирным шрифтом значений ста ячеек по диагонали активного листа:

```
For i = 1 To 100  
    Set Cell(i)=Cells(i,i)  
    Cell(i).Font.Bold = True  
Next i
```

Размерность массива

Массив Cell, описанный в предыдущем разделе, является одномерным. Но можно определить и массивы, размерность которых превышает единицу. Например, массив

```
Dim Cell(1 To 10, 1 To 100) As Range
```

является двумерным. Его первый индекс изменяется от 1 до 10, а второй — от 1 до 100. Соответственно, количество элементов массива составляет $10 \times 100 = 1000$.

Массивы VBA могут иметь до 60 измерений. Однако массивы большой размерности практически никогда не используются, так как для их обработки требуется слишком много оперативной памяти.

Динамические массивы

Когда массив описывается следующим образом:

```
Dim FileName(1 To 10) As String
```

его размер фиксирован, потому что верхняя и нижняя границы индексов указаны явно. Однако возникают ситуации, когда во время описания переменных еще неизвестно, массив какого размера вам нужен. Для этих случаев в языке VBA существуют динамический массив и оператор ReDim.

Динамический массив описывается с помощью пустых скобок:

```
Dim FileName() As String
```

Впоследствии можно явно указать размер этого массива. Это делается следующим образом:

```
ReDim FileName(1 to 10)
```

Этот оператор также позволяет изменить размер уже существующего массива:

```
ReDim FileName(1 to 100)
```

Имейте в виду, что изменение размера массива приводит к потере содержащихся в нем данных. Избежать этого можно с помощью оператора Preserve:

```
ReDim Preserve FileName(1 to 200)
```

Однако в данном случае вы можете менять только верхнюю границу массива и только последнее измерение в многомерном массиве.

Функция UBound

Функция UBound используется для определения верхней границы массива. Она очень полезна, когда требуется изменить его размерность. Предположим, что нужно собрать неизвестное число имен файлов в массив с именем FileName. Следующий код проверяет, не является ли верхний индекс массива меньшим, чем номер файла iNextFile, и в случае положительного результата увеличивает число индексов массива на 10, предоставляя пространство для новых имен:

```
If UBound(FileName) < iNextFile Then  
    ReDim Preserve FileName(UBound(FileName) + 10)  
End If
```

При изменении числа элементов массива имеет смысл добавлять значительное количество новых индексов, чтобы уменьшить количество изменений. Именно поэтому в вышеприведенном примере прибавляется 10, а не 1. Существует компромисс между тем, сколько раз меняется число элементов, и пространством, которое можно потерять, не до конца задействовав измененный массив.

Стандарты именования

Программы на языке VBA иногда бывают настолько сложными, что хороши любые средства, позволяющие сделать их более читаемыми. Кроме того, с течением времени идеи, лежащие в основе программы, забываются, и попытаться определить смысл кода можно только по его виду. Именно поэтому так важно добавлять в программу содержательные комментарии.

Другим способом улучшения читаемости программ является выработка стандарта именования констант, переменных, процедур и прочих элементов. В общем случае имя должно обладать двумя свойствами. Во-первых, напоминать о функции переменной. Предположим, требуется сопоставить переменные типа Chart с рядом диаграмм. Следующий код

```
Dim chrt1 As Chart, chrt2 As Chart  
Set chrt1 = Charts("Продажи")  
Set chrt2 = Charts("Транзакции")
```

не содержит ошибок, но если он содержится в программе длиной в тысячу строк, вряд ли через шесть месяцев можно будет вспомнить, что значат переменные chrt1 и chrt2. Имеет смысл давать этим переменным значимые имена:

```
Dim chrtSales As Chart, chrtTrans As Chart  
Set chrtSales = Charts("Продажи")  
Set chrtTrans = Charts("Транзакции")
```

Конечно, из любого правила существуют исключения, но в общем случае всем элементам программы (константы, переменные, процедуры, формы, элементы управления и т. п.) лучше присваивать значимые имена.

Во-вторых, имена переменных должны отражать некоторые их свойства, например, принадлежность к определенному типу данных. Многие программисты используют стандарт, по которому первые несколько букв переменной указывают на ее тип данных. Иногда это называют венгерским стандартом, так как он был разработан венгерским программистом Чарльзом Симонаи.

В табл. 5.3 и 5.4 описаны условные обозначения, которые используются для обычных и объектных переменных соответственно. Разумеется, вы можете разра-

ботать собственную систему, главное, чтобы она была достаточно последовательной.

Таблица 5.3. Стандарты именования для обычных переменных

Переменная	Приставка
Boolean	b или f
Byte	b или bt
Currency	cur
Date	dt
Double	d или dbl
Integer	i, c или int
Long	l, c или lng
Single	s или sng
String	s или str
Тип, определенный пользователем	u или ut
Variant	v или var

Приставка «с» обычно используется для переменных, которые принимают участие в вычислениях.

Таблица 5.4. Стандарты именования для объектных переменных

Переменная	Приставка
Chart	ch или chrt
Workbook	wb или wbk
Worksheet	ws или wks
Pivot Table	pt или pvt
Font	fnt
Range	rng

Кроме принадлежности к определенному типу данных каждая переменная имеет *область видимости (scope)* и *время жизни (lifetime)*. Некоторые программисты включают в имена переменных букву g для обозначения уровня проекта и букву m для обозначения уровня модуля. Например, переменная giSize представляет собой глобальную переменную типа Integer.

Область видимости переменных

Переменные и константы имеют область видимости, в которой к ним возможен доступ. Существуют переменные уровня процедуры (их также называют локальными), модуля и проекта. То же самое можно сказать о константах. Правила определения области видимости на первый взгляд выглядят запутанными, но позднее вы убедитесь, что они имеют смысл.

Переменные уровня процедуры

Переменная доступна на уровне процедуры, если она объявлена внутри этой процедуры ключевым словом Dim. Сюда относятся переменная LocalVar и константа

LocalConstant на рис. 5.2. Попытка использовать эти переменные и константы вне процедуры, в которой они были определены, приведет к ошибке. То есть если попытаться запустить процедуру B, вы получите сообщение «variable not defined», а имя переменной LocalVar будет выделено.

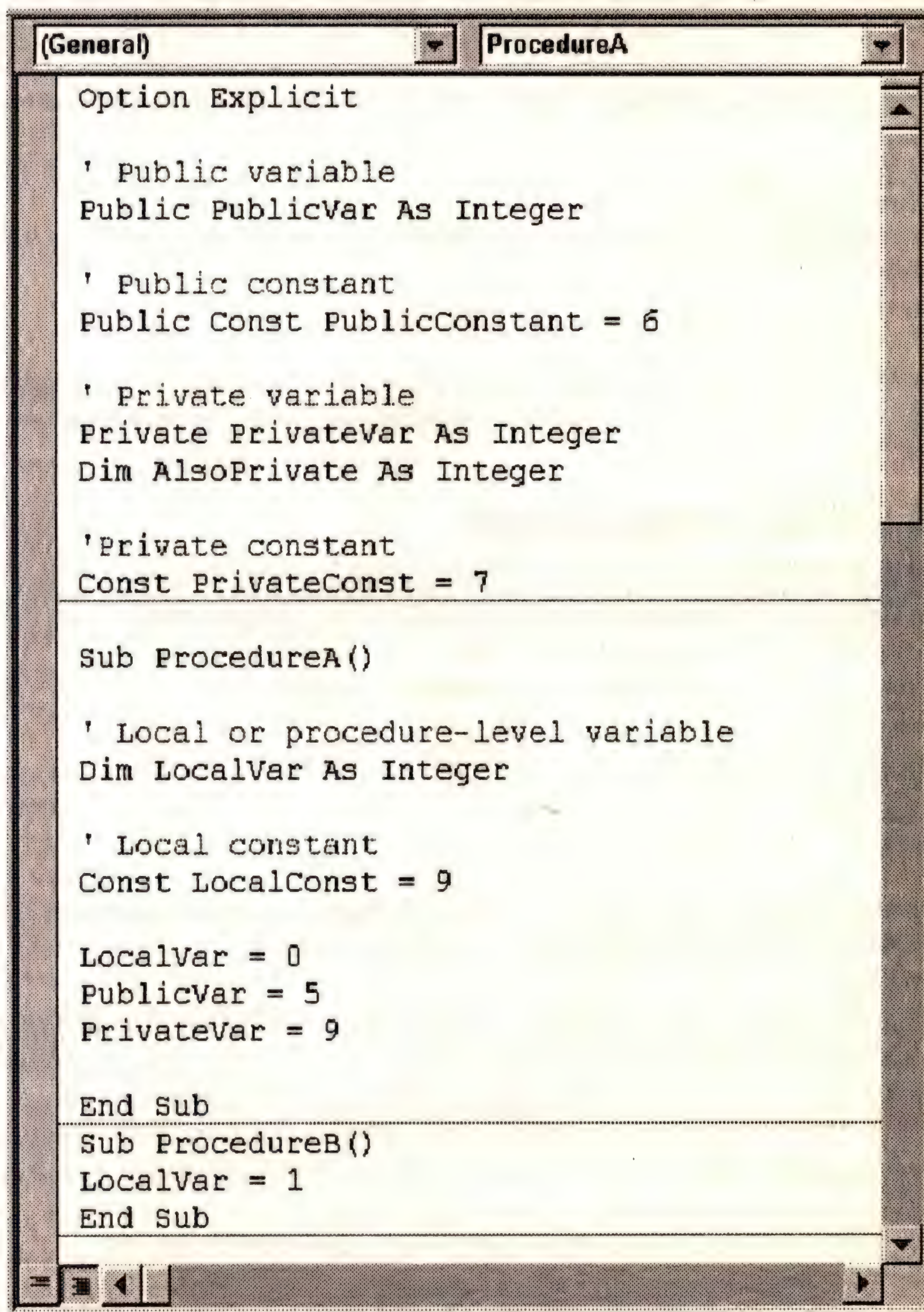


Рис. 5.2. Пример областей видимости переменных

Одним из основных преимуществ переменных уровня процедуры является возможность повторного использования одних и тех же имен в различных процедурах.

Переменные уровня модуля и проекта

Переменные уровня модуля и проекта определяются в разделе описаний любого модуля (стандартного, модуля классов или UserForm).

Переменная уровня проекта доступна в процедурах всех модулей вне зависимости от того, где она была изначально описана, в то время как переменная уровня модуля доступна только для процедур того модуля, в котором она была описана.

Переменные уровня проекта описываются с помощью ключевого слова `Public`. Например:

```
Public APubInt As Integer  
Public Const APubConst = 7
```

Переменные уровня модуля описываются с помощью ключевого слова `Private`. Например:

```
Private APrivateInt As Integer  
Private Const APrivateConst = 7
```

Ключевое слово `Dim` на уровне модуля имеет то же самое значение, что и слово `Private`. Но так как оно не столь ясно описывает ситуации, использовать его нежелательно.

Время жизни переменной

Разница между временем жизни и областью видимости достаточно проста. Время жизни показывает, как долго переменную можно использовать, а область видимости определяет, где ее можно использовать.

Для иллюстрации рассмотрим следующую процедуру:

```
Sub ProcedureA()  
    Dim LocalVar As Integer  
    LocalVar = 0  
    Call ProcedureB  
    LocalVar = 1  
End Sub
```

Обратите внимание на то, что переменная `LocalVar` является переменной уровня процедуры. Соответственно, при выполнении строки

```
Call ProcedureB
```

происходит переход к процедуре В, в которой переменная `LocalVar` не определена. Но она до сих пор действительна. Другими словами, данная переменная существует и имеет некоторое значение, просто она недоступна для кода процедуры В. При этом внутри данной процедуры также может быть определена переменная `LocalVar`, которая не имеет ничего общего с переменной, носящей это имя в процедуре А.

После завершения процедуры В управление переходит к строке

```
LocalVar = 1
```

что вполне правомерно, так как мы снова вернулись в процедуру А.

Соответственно, время жизни переменной `LocalVar` продолжается с момента начала выполнения процедуры А до момента ее завершения, включая период выполнения процедуры В, в течение которого с указанной переменной ничего не происходило.

Статические переменные

Как вы уже убедились, переменные сохраняют свое значение с течением времени, вне зависимости от того, являлись ли они действительными все время или нет. Однако после окончания времени жизни значение переменной теряется. Рассмотрим следующую процедуру:

```
Sub ProcedureA()  
    Call ProcedureB  
    Call ProcedureB  
    Call ProcedureB  
    Call ProcedureB  
    Call ProcedureB  
End Sub  
Sub ProcedureB()  
    Dim x As Integer  
    x = 5  
...  
End Sub
```

Сущность процедуры А состоит в пятикратном вызове процедуры В. При этом каждый раз заново создается переменная *x*, которая исчезает к концу процедуры. И этот процесс повторяется пять раз.

Впрочем, нам именно это и нужно. Однако бывают ситуации, когда требуется, чтобы переменная существовала дольше процедуры, в которой она определена. Предположим, что действия, которые процедура должна выполнять в первый раз, не имеет смысла повторять снова и снова. Например, следующая процедура меняет шрифт в выделенных ячейках на Comic Sans:

```
Sub ToComic()  
    Selected.Font.Name = "Comic Sans"  
End Sub
```

Данный шрифт выглядит крайне неофициально, поэтому имеет смысл узнать у пользователя, уверен ли он в необходимости его применения. Однако вряд ли стоит повторять этот вопрос при каждом вызове процедуры. То есть требуется переменная «с памятью», которая отслеживает номер вызова процедуры.

Статическими переменными (static variable) называются локальные переменные, время жизни которых заканчивается только после завершения работы всех процедур, включая ту, в которой эта переменная была определена. В действительности они сохраняют свое значение, пока остается активным документ или шаблон, содержащий модуль кода, даже если выполнения кода не происходит.

То есть статическая переменная имеет область видимости локальной переменной, а время жизни — как у переменной уровня модуля.

Рассмотрим вариант вышеприведенного кода в листинге 5.2. Сначала описывается статическая логическая переменная *NotFirstTime*. Может показаться, что проще в данном случае использовать переменную *FirstTime*, но существует небольшая проблема, а именно по умолчанию логическим переменным присваивается значение *False*, соответственно при первом запуске процедуры переменная *FirstTime* будет иметь именно это значение, что не соответствует нашим пожеланиям.

Листинг 5.2. Вариант процедуры ToComic со статической переменной

```
Sub ToComic()  
    ' Описание статической логической переменной
```

продолжение ➤

Листинг 5.2 (продолжение)

```
Static NotFirstTime As Boolean
```

```
' При первом запуске процедуры появляется окно с вопросом
```

```
If NotFirstTime = False Then
```

```
    If MsgBox("Comic Sans is a bit informal. Proceed?", _
```

```
        vbYesNo) = vbYes Then
```

```
        ' Изменение шрифта
```

```
        Selection.Font.Name = "Comic Sans MS"
```

```
End If
```

```
' Если процедура вызывается не в первый раз
```

```
NotFirstTime = True
```

```
Else
```

```
' При повторном вызове процедуры шрифт изменяется по умолчанию
```

```
Selection.Font.Name = "Comic Sans MS"
```

```
End If
```

```
End Sub
```

С помощью оператора If проверяется значение переменной NotFirstTime. Если оно равно False, значит, процедура выполняется впервые и требуется вызвать диалоговое окно, содержащее текст Comic Sans is a bit informal. Proceed? (Шрифт Comic Sans выглядит неофициально. Продолжить?) (рис. 5.3).

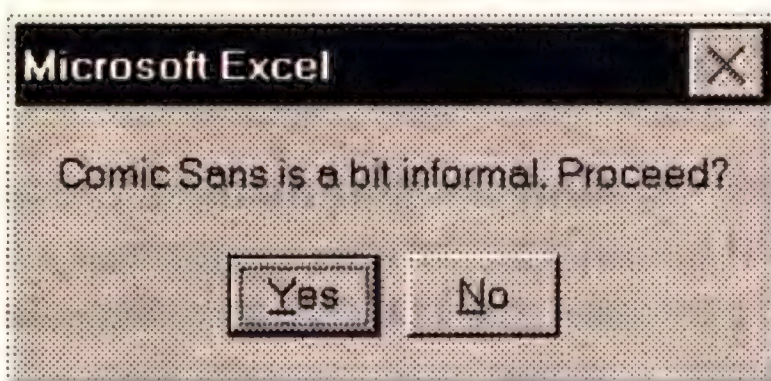


Рис. 5.3. Диалоговое окно, появляющееся, если статическая переменная NotFirstTime имеет значение False

Если пользователь щелкает на кнопке Yes, шрифт изменяется. В противном случае переменная NotFirstTime принимает значение True. Так как она является статической, это значение будет действительным даже после окончания подпрограммы.

При следующем запуске этой подпрограммы переменная NotFirstTime будет иметь значение True и условие

```
If NotFirstTime = False Then
```

не будет выполнено. Соответственно, окно с предупреждением не появится. Вместо этого произойдет автоматическая смена шрифта.

Статические переменные используются не очень часто, но в некоторых случаях без них просто не обойтись.

Может показаться, что эта задача выполняется с помощью переменной уровня процедуры. Однако хорошим стилем считается использование переменных с наименьшей областью видимости. В данном случае это переменная уровня процеду-

ры с «растянутым» временем жизни. Это предотвращает возможность случайного изменения переменной в других частях кода. Имейте в виду, что упомянутый код может являться частью огромной программы, поэтому лучше сделать переменную `NotFirstTime` невидимой вне процедуры, в которой она была определена.

Инициализация переменных

Начало выполнения процедуры сопровождается инициализацией определенных в ней переменных, то есть присвоением им неких начальных значений. Но в общем случае не стоит полагаться на инициализацию, так как программа становится менее читаемой и увеличивается вероятность логических ошибок. Лучше в явном виде указывать значение всех переменных, как это сделано в следующем примере:

```
Sub Example()
```

```
Dim x As Integer  
Dim s As String
```

```
x = 0 ' Переменной x присвоено значение 0  
s = "" ' Переменной s присвоена пустая строка
```

```
' Остальной код...
```

```
End Sub
```

Имейте в виду, что статические переменные невозможно инициализировать. Важно помнить следующие правила инициализации переменных:

- численным переменным (типа `Integer`, `Long`, `Single`, `Double` и `Currency`) обычно присваивается значение 0;
- строкам с переменной длиной в качестве начального значения обычно присваивается пустая строка;
- строки с фиксированной длиной обычно заполняются символами, код которых в ASCII-таблице соответствует 0;
- переменным типа `Variant` присваивается значение `Empty`;
- объектным переменным присваивается значение `Nothing`.

Ключевое слово `Nothing` в Excel имеет ряд функций. Как вы узнаете в главе 8, оно используется для освобождения объектных переменных. Если же оно является возвращенным значением некоей функции, это обычно значит, что какая-то из операций не была завершена. Наконец, именно она применяется для инициализации объектных переменных.

Операторы VBA

Набор основных операторов языка VBA показан в табл. 5.5.

Оператор `Mod` возвращает остаток при операции деления. Например, операция `8 Mod 3` возвращает значение 2.

Таблица 5.5. Операторы языка VBA

Тип	Название	Символ
Арифметические операторы	Сложение	+
	Вычитание	-
	Умножение	*
	Деление	/
	Деление нацело	\
	Возведение в степень	^
	Остаток от деления	Mod
Строковый оператор	Объединение	&
Логические операторы	AND	AND
	OR	OR
	NOT	NOT
Операторы сравнения	Равно	=
	Меньше, чем	<
	Больше, чем	>
	Меньше или равно	<=
	Больше или равно	>=
	Не равно	<>

Для иллюстрации объединения строк рассмотрим выражение:
"To be or" & "not to be"
В результате получается выражение To be or not to be (Быть или не быть).

6

Функции и подпрограммы

Как вы уже знаете, в языке VBA существует два вида процедур: функции и подпрограммы. Напомним, что единственное отличие между ними состоит в том, что функция возвращает некое значение, а подпрограмма — нет.

Вызов функций

Описание функции имеет следующий вид:

```
[Public или Private] Function Имя функции(Параметр1 As Тип данных, _  
Параметр2 As Тип данных, ...) As Тип возвращаемого значения
```

Обратите внимание на то, что тип данных указывается не только для всех параметров функции, но и для возвращаемого значения. Не описанные в явном виде переменные будут принадлежать к типу Variant.

Значение ключевых слов Public и Private мы обсудим чуть позже, но, вероятно, вы уже догадались, что они указывают на область видимости функции.

В качестве примера рассмотрим функцию, увеличивающую значение исходной переменной Value на единицу.

Листинг 6.1. Функция AddOne

```
Public Function AddOne(Value As Integer) As Integer  
    AddOne = Value + 1  
End Function
```

Чтобы использовать возвращаемое значение функции, достаточно поместить вызов функции в том месте выражения, где должно располагаться это значение. Например, код

```
MsgBox " Adding 1 to 5 gives: " & AddOne(5)
```

вызывает диалоговое окно с текстом Adding 1 to 5 gives: 6 (Если к 5 прибавить 1, получим: 6), показанное на рис. 6.1. В данном случае выражение AddOne(5) заменяется возвращаемым значением данной функции, которое равно 6.

Помните, что любые параметры функции при ее вызове должны быть заключены в скобки.

Для получения значения функции нужно назначить ее имя возвращаемому значению внутри ее тела. Более сложная функция показана в листинге 6.2.

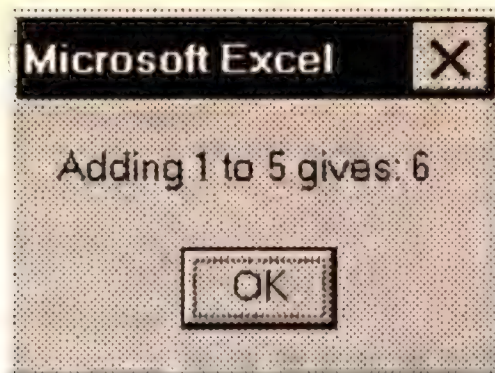


Рис. 6.1. Диалоговое окно с сообщением из листинга 6.1

Листинг 6.2. Назначение возвращаемого значения функции

```
Function ReturnCount() As Variant
```

```
    ' Вычисление количества выделенных ячеек
```

```
    If TypeName(Selection) = "Range" Then
```

```
        ReturnCount = Selection.Count
```

```
    Else
```

```
        ReturnCount = "Не применимо"
```

```
    End If
```

```
End Function
```

Эта функция показывает количество выделенных ячеек. При выделении объекта другого типа, например диаграммы, функция возвращает значение "Не применимо". Тип Variant в данном случае использован потому, что возвращаемым значением может быть как число, так и строка. Обратите внимание: в теле функции ReturnCount ей присваиваются два разных значения. Но возвращаемое значение зависит от функции TypeName(Selection). Так как значения последней являются взаимоисключающими, при каждом вызове функции ReturnCount возвращается только одно из них.

Из-за того что функция должна возвращать некое значение, ее невозможно напрямую вызвать в диалоговом окне Macro (Макрос), которое появляется по команде Tools ► Macro ► Macros (Сервис ► Макрос ► Макросы), а также сопоставить с кнопкой панели инструментов или командой меню в пользовательском интерфейсе Excel. Для вызова функции ее требуется включить в подпрограмму.

Вызов подпрограмм

Описание подпрограммы выглядит следующим образом:

```
[Public или Private] Sub Имя подпрограммы (Параметр1 As Тип данных, _  
    Параметр2 As Тип данных, ...)
```

Единственным отличием от описания функции является отсутствие возвращаемого значения. Обратите также внимание на то, что вместо ключевого слова Function используется слово Sub.

Подпрограммы нельзя использовать в выражениях, так как они не возвращают никаких значений. Для вызова подпрограммы SubroutineA можно написать:

```
Call SubroutineA(Параметры, ...)
```

или просто

```
SubroutineA Параметры, ...
```


Параметры заключаются в скобки только в том случае, когда используется ключевое слово `Call`.

Параметры и аргументы

Рассмотрим простую подпрограмму, вызывающую окно сообщения с именем пользователя:

```
Sub DisplayName(sName As String)
    MsgBox "Меня зовут " & sName
End Sub
```

Для вызова этой подпрограммы можно написать, к примеру:

```
DisplayName "Маша"
```

или

```
Call DisplayName("Маша")
```

Переменная `sName` в описании процедуры

```
Sub DisplayName(sName As String)
```

называется *параметром* (*parameter*). Вызов процедуры должен содержать строковую переменную, которая будет соответствовать переменной `sName`. Значение, используемое вместо параметра при вызове процедуры, называется *аргументом* (*argument*). В предыдущем примере аргументом являлась строка "Маша".

Иногда программисты не различают понятия параметров и аргументов, используя их в различных случаях. Однако так как параметр подобен переменной, а аргумент подобен ее значению, путать их — все равно что путать переменную и ее значение.

Необязательные аргументы

В языке VBA аргументы процедуры могут быть сделаны необязательными с помощью ключевого слова `Optional`. Для иллюстрации рассмотрим листинг 6.3, в котором процедура меняет размер букв и шрифт в выделенных ячейках.

Листинг 6.3. Использование необязательного аргумента

```
Sub ChangeFormatting(FontName As String, _
    Optional FontSize As Variant)
```

```
    ' Изменение шрифта
```

```
    Selection.Font.Name = FontName
```

```
    ' Изменение размера букв при наличии аргумента
```

```
    If Not IsMissing(FontSize) Then
```

```
        Selection.Font.Size = CInt(FontSize)
```

```
    End If
```

```
End Sub
```

Второй параметр описан с помощью ключевого слова `Optional`. Это дает возможность вызывать процедуры как с аргументом для этого параметра, так и без него. Например:

```
Change.Formatting("Arial Narrow", 24)
```


и

```
Change.Formatting("Arial Narrow")
```

В теле процедуры используется функция `IsMissing`, которая проверяет наличие аргумента. Если аргумент присутствует, размер шрифта изменяется. Параметр `Font.Size` мы отнесли к типу `Variant`, потому что функция `IsMissing`, к сожалению, работает только с параметрами этого типа. Впоследствии переменная типа `Variant` преобразуется к типу `Integer` с помощью функции `CInt`.

Процедура может иметь произвольное число необязательных аргументов, но все они должны перечисляться в конце списка параметров. Вот пример неправильного описания:

```
Sub ChangeFormatting(Optional FontName As String, FontSize As Single)
```

Если пропустить необязательный аргумент в центре списка, при вызове процедуры потребуется использовать дополнительный пробел. Например, если процедура описывается следующим образом

```
Sub ChangeFormatting(Optional FontName As String, _
                    Optional FontSize As Single, _
                    Optional FontBold As Boolean)
```

Вызов процедуры, которая меняет шрифт в ячейке на Arial и выделяет написанный текст полужирным **стилем**, выглядит так:

```
ChangeFormat "Arial". . True
```

Чтобы избежать недоразумений, сразу подчеркнем, что некоторые встроенные процедуры Excel имеют необязательные аргументы, а некоторые — нет. Разумеется, пропускать аргументы можно только в случае, когда в документации или в описании процедуры ясно сказано, что они являются необязательными.

Именованные аргументы

Некоторые процедуры языка VBA содержат большое число параметров. Вот пример описания одной из форм функции `SaveAs`:

```
SaveAs (FileName As String, FileFormat As Variant, Password As Variant, _
WriteResPassword As Variant, ReadOnlyRecommended As Variant, _
CreateBackup As Variant, AddToMru As Variant, TextCodepage As _
Variant, TextVisualLayout As Variant),
```

где все параметры являются необязательными. А вот пример вызова этой процедуры:

```
SaveAs "c:\temp\text.xls". . . . . True . . . True
```

Его непросто понять, не так ли?

Аргументы, указанные при вызове процедуры, называются *позиционными аргументами* (*positional arguments*), так как их позиция указывает, какие параметры они заменяют. Именно поэтому требуется использовать пробел для обозначения отсутствующих аргументов.

Впрочем, в языке VBA также существуют *именованные аргументы* (*named arguments*), при использовании которых вызов предыдущей процедуры выглядит следующим образом:

```
SaveAs FileName:="c:\temp\test.xls". _
CreateBackup:=True. _
AddToMru:=True
```


Обратите внимание на специальный синтаксис, в частности на использование двоеточия перед знаком равенства.

Это более удобный способ вызова функции. В общем случае в качестве преимуществ именованных аргументов перед аргументами положения можно указать:

- увеличение читаемости программы;
- для обозначения недостающих аргументов больше не требуется использовать пробелы;
- порядок аргументов перестает иметь значение.

Так как именованные аргументы делают программу более читаемой, я настоятельно рекомендую применять именно их. Однако так как для них требуется значительно больше места, в коротких примерах данной книги они использоваться не будут.

Передача аргументов

Передача аргументов из одной программы в другую осуществляется двумя способами — *по ссылке (by reference)* и *по значению (by value)*. Выбор способа передачи осуществляется с помощью ключевых слов `ByRef` и `ByVal`.

Для иллюстрации разницы рассмотрим процедуру в листинге 6.4. Подпрограмма `ProcedureA` присваивает переменной уровня модуля `x` значение 5, показывает это значение, затем вызывает подпрограмму `AddOne` с аргументом `x` и снова отображает значение этой переменной.

Листинг 6.4. Использование ключевых слов `ByRef` и `ByVal`

```
Sub ProcedureA()
    x = 5 ' Присваиваем переменной x значение 5
    MsgBox x ' Показываем значение x
    Call AddOne(x) ' Вызываем процедуру AddOne
    MsgBox(x) ' Снова показываем значение x
End Sub
```

```
Sub AddOne(ByRef i As Integer)
    i = i + 1
End Sub
```

Обратите внимание на наличие ключевого слова `ByRef` в описании подпрограммы `AddOne`. Это ключевое слово указывает на то, что нужно передать ссылку на переменную `x` в подпрограмму `AddOne`. Следовательно, параметр `i` заменяется параметром `x`. То есть строка

```
i = i + 1
```

заменяется строкой

```
x = x + 1
```

В результате после вызова подпрограммы `AddOne` переменная `x` будет иметь значение 6.

А теперь представим, что в подпрограмме `AddOne` ключевое слово `ByRef` заменено ключевым словом `ByVal`. В этом случае будет передана не ссылка на переменную `x`, а ее значение в конкретный момент. То есть переменная `i` просто примет значение 5. После прибавления единицы значение этой переменной станет равным 6.

Но значение переменной *x* при этом не изменится, то есть окно сообщения оба раза покажет значение 5.

Оба ключевых слова имеют свои области применения. Если нужно изменить значение аргумента, соответствующий параметр описывается с ключевым словом *ByRef*. Это дает вызываемой процедуре доступ к аргументу, что вы видели на примере процедуры, написанной в начале данного раздела. В противном случае подпрограмма *AddOne* совершенно бесполезна, так как единица прибавляется к локальной переменной *i*, которая исчезает сразу после окончания процедуры.

С другой стороны, если требуется использовать аргумент внутри подпрограммы, не меняя его значения, он описывается с ключевым словом *ByVal*. В результате вызванная подпрограмма получит только значение аргумента.

Для иллюстрации рассмотрим листинг 6.5. Процедура *ProcedureA* применяет функцию *CountCharacters* к тексту в первой ячейке активного листа. Полученное в результате число букв затем показывается в окне сообщения.

Листинг 6.5. Передача аргумента по значению

```
Sub ProcedureA()  
    Dim sText As String  
    sText = ActiveSheet.Cells(1,1).Text  
    MsgBox CountCharacters(sText)  
End Sub  
  
Function CountCharacters(ByVal sTxt As String)  
    CountCharacters = Len(sTxt)  
End Function
```

В данном случае функция *CountCharacters* не должна изменять текст, она только подсчитывает количество символов. Именно поэтому используется передача аргумента по значению. В результате переменная *sTxt* получит значение переменной *sText*, то есть копию текста.

Представьте себе, что функция *CountCharacters* заменена процедурой длиной в сотню или даже тысячу строк и вы не знаете, насколько надежным является это творение чужих рук. Разумеется, вряд ли вы захотите, чтобы эта процедура меняла ваш текст. Но вместо проверки кода на наличие ошибок достаточно использовать передачу аргумента *sTxt* по значению. В результате процедура сомнительной надежности будет работать с копией текста, оставив без изменений оригинал.

Передача аргумента по значению имеет один недостаток. Иногда для этого требуется слишком много оперативной памяти, и, следовательно, время выполнения процедуры сильно увеличивается. Скажем, в предыдущем примере требуется сделать копию текста, которая будет передана в параметр *sTxt*.

Итак, какой вывод следует из написанного выше? Если требуется, чтобы процедура меняла аргумент, его следует передавать по ссылке. Если же аргумент желательно сохранить без изменений, его нужно передать по значению. Исключением является случай, когда снижение производительности слишком велико или если вы уверены в том, что непредумышленного изменения аргумента не произойдет.

При отсутствии в описании аргументов ключевых слов *ByRef* и *ByVal* передача осуществляется по ссылке. То есть аргумент по умолчанию меняется вызванной процедурой, если вы не укажете противного.

Выход из процедуры

В языке VBA существуют операторы Exit Sub и Exit Function, позволяющие выйти из процедуры до ее завершения. В листинге 6.6, если параметр принимает неподходящее значение, появляется окно с предупреждением и процедура завершается.

Листинг 6.6. Использование оператора Exit Sub

```
Sub DisplayName(sName As String)
    If sName = "" then
        MsgBox "Пожалуйста, введите имя."
        Exit Sub
    End If
    MsgBox "Введенное имя " & sName
End Sub
```

Область видимости процедур

Подобно переменным и константам, процедуры имеют область видимости. Она указывается при описании процедуры с помощью ключевых слов Private и Public. Например:

```
Public Function AddOne(i As Integer) As Integer
```

или

```
Private Function AddOne(i As Integer) As Integer
```

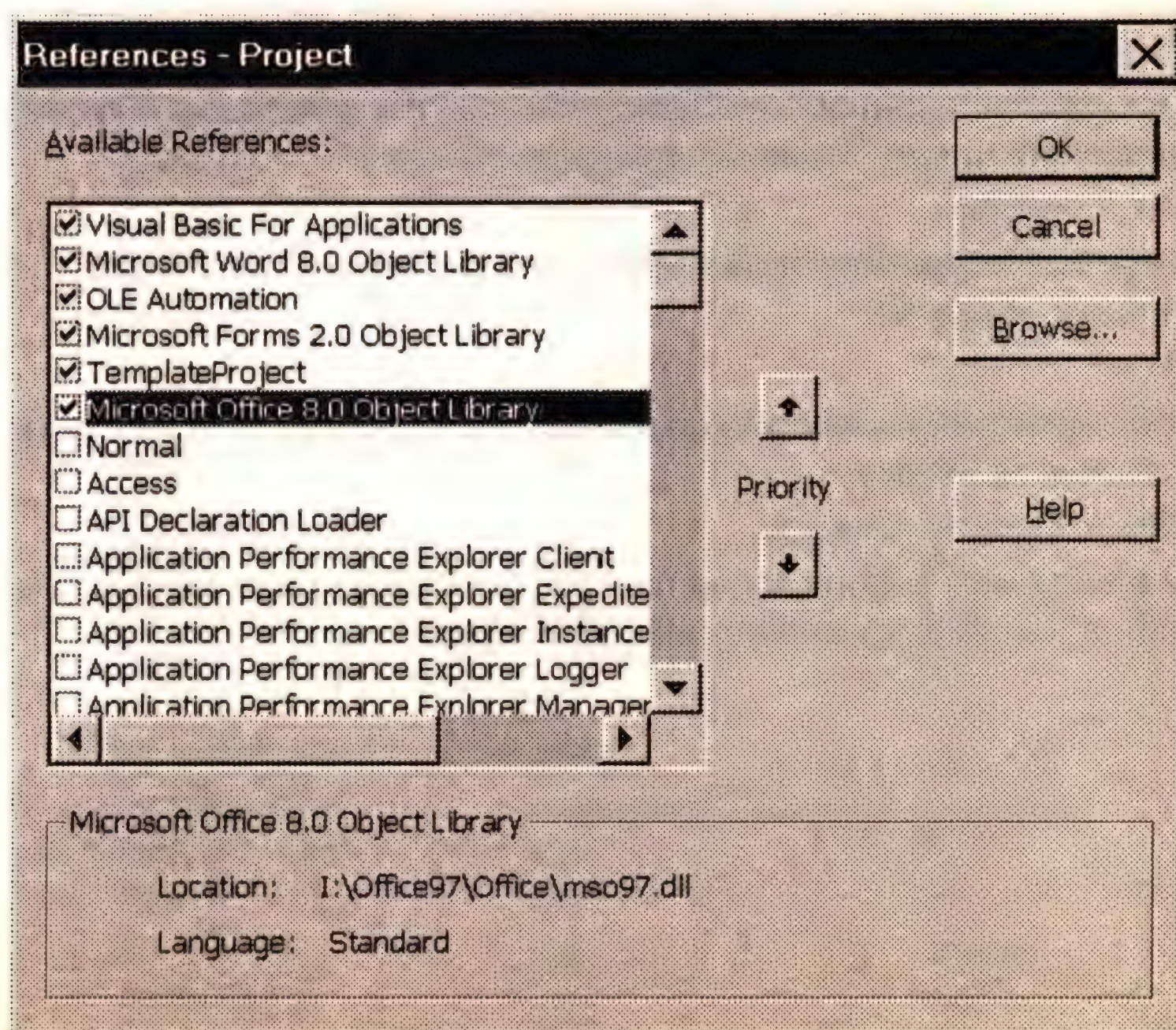


Рис. 6.2. Диалоговое окно References

Разница между ними проста. Если в описании процедуры присутствует ключевое слово `Private`, ее вызов может быть осуществлен только внутри модуля, в котором она определена. Это так называемые процедуры уровня модуля. Ключевым словом `Public` описывают процедуры уровня проекта, которые можно вызвать в любом из модулей.

Отсутствие в описании процедуры ключевых слов `Private` и `Public` означает, что это процедура уровня модуля.

Ссылки на проект

Существует возможность использовать процедуры уровня проекта из других программ.

Обычно проект, связанный с книгой, использует только определенные в нем процедуры. Собственно говоря, использование в одном проекте процедур из другого является не очень хорошим стилем программирования. Но в некоторых случаях без этого не обойтись. Чтобы добавить ссылку на проект, используйте диалоговое окно `References`, показанное на рис. 6.2. Оно вызывается командой `References` меню `Tools` редактора VBA.

Уточненное имя процедуры

При вызове процедуры уровня модуля существует потенциальная проблема, так как в различных модулях могут находиться процедуры с одинаковыми именами. В этом случае будет выполнена первая из обнаруженных процедур, но не факт, что именно она требуется в данном случае.

Чтобы избежать подобных проблем, используйте *уточненные имена процедур* (*qualified procedure name*). Такие имена имеют форму:

Имя_модуля.Имя_процедуры

Например, вызов процедуры `AddOne`, описанной в модуле `Utilities`, будет выглядеть следующим образом:

`Utilities.AddOne`

При необходимости можно указать также имя проекта. В этом случае имя процедуры будет иметь форму:

Имя_проекта.Имя_модуля.Имя_процедуры

Имейте в виду, что в данном случае нужно использовать именно имя проекта, а не имя файла. По умолчанию это имя `VBAProject`.

7

Встроенные функции и операторы

В языке VBA существует множество встроенных функций и операторов. Они перечислены в табл. 7.1. В этой и следующей главах будут рассмотрены те из них, которые используются чаще всего.

Для упрощения описания мы используем принятые в Microsoft обозначения, например, для обозначения необязательных параметров будем применять квадратные скобки. Скажем, в следующей процедуре второй параметр является необязательным:

```
Sub ChangeFormat(FontName [, FontSize])
```

Обратите внимание на то, что в данном случае не указан тип данных. Впрочем, об этом мы поговорим позднее.

Таблица 7.1. Функции VBA

Abs	DoEvents	IsEmpty	Right
Array	Environ	IsError	RightB
Asc	EOF	IsMissing	Rnd
AscB	Error	IsNull	Round*
AscW	Exp	IsNumeric	RTrim
Atn	FileAttr	IsObject	Second
CallByName*	FileDateTime	Join*	Seek
Cbool	FileLen	Lbound	Sgn
Cbyte	Filter*	Lcase	Shell
Ccur	Fix	Left	Sin
Cdate	Format	LeftB	SLN
CDbl	FormatCurrency*	Len	Space
Cdec	FormatDateTime*	LenB	Spc
Choose	FormatNumber*	LoadPicture	Split*
Chr	FormatPercent*	Loc	Sqr
ChrB	FreeFile	LOF	Str
ChrW	FV	Log	StrComp
Cint	GetAllSettings	Ltrim	StrConv
CLng	GetAttr	Mid	String

продолжение ↗

Таблица 7.1 (продолжение)

Command	GetAutoServerSettings	MidB	StrReverse*
Cos	GetObject	Minute	Switch
CreateObject	GetSettings	MIRR	SYD
CSng	Hex	Month	Tab
CStr	Hour	MonthName*	Tan
CurDir	Iif	MsgBox	Time
Cvar	IMEStatus	Now	Timer
CVDate	Input	Nper	TimeSerial
CVErr	InputB	NPV	TimeValue
Date	InputBox	Oct	Trim
DateAdd	InStr	Partition	TypeName
DateDiff	InStrB	Pmt	UBound
DatePart	InstrRev*	PPmt	UCase
DateSerial	Int	PV	Val
DateValue	Ipmt	QBColor	VarType
Day	IRR	Rate	WeekdayWeekdayname*
DDB	IsArray	Replace*	Year
Dir	Isdate	RGB	

* — функции, помеченные этим значком, существуют только в пакете Office 9/10

Функция MsgBox

Вы уже не раз сталкивались с функцией MsgBox в примерах из предыдущих глав. Теперь поговорим о ней более подробно. Она используется для вывода на экран информационного окна, которое исчезает после того, как пользователь нажмет кнопку. Чаще всего описание этой функции выглядит следующим образом:

```
MsgBox(prompt [. buttons] [. title])
```

Это отнюдь не полное описание синтаксиса. Существуют необязательные параметры, связанные с контекстом подсказки, о которых вы можете узнать в справочной документации.

Prompt — параметр типа String, содержащий сообщение, которое показывается в информационном окне. Для создания сообщения из нескольких строк используется константа vbCrLf.

Buttons — параметр типа Long, содержащий сумму значений, которые определяют различные свойства информационного окна. Это число и тип показываемых кнопок, стиль значков, идентичность заданных по умолчанию кнопок и модальность окна сообщения. *Модальное диалоговое окно (system modal dialog box)* появляется поверх всех окон, открытых в данный момент, и именно в его поля осуществляется ввод данных, в то время как *режимное диалоговое окно (application modal dialog box)* появляется только поверх окна приложения. Различные значения параметра Buttons показаны в табл. 7.2. Они же формируют перечисление VbMsgBoxStyle.

Например, код
MsgBox "Proceed?", vbQuestion + vbYesNo
приведет к появлению информационного окна, которое содержит знак вопроса, надпись Proceed? (Продолжать?) и две кнопки Yes (Да) и No (Нет). Его вид показан на рис. 7.1.

Таблица 7.2. Значение аргумента Buttons функции MsgBox

Назначение	Константа	Значение	Описание
Типы кнопок	VbOKOnly	0	Появляется только кнопка OK
	VbOKCancel	1	Появляются кнопки OK и Cancel
	vbAbortRetryIgnore	2	Появляются кнопки Abort, Retry и Ignore
	vbYesNoCancel	3	Появляются кнопки Yes, No и Cancel
	VbYesNo	4	Появляются кнопки Yes и No
	vbRetryCancel	5	Появляются кнопки Retry и Cancel
	VbCritical	16	Показывается значок критического сообщения
Типы значков	VbQuestion	32	Показывается вопросительный знак
	vbExclamation	48	Показывается знак предупреждающего сообщения
	vbInformation	64	Показывается знак информационного сообщения
Кнопки по умолчанию	vbDefaultButton1	0	По умолчанию берется первая кнопка
	vbDefaultButton2	256	По умолчанию берется вторая кнопка
	vbDefaultButton3	512	По умолчанию берется третья кнопка
	vbDefaultButton4	768	По умолчанию берется четвертая кнопка
Модальность	vbApplicationModal	0	Появляется режимное окно диалога
	vbSystemModal	4096	Появляется модальное окно диалога

Параметр title представляет собой строковое выражение, показываемое в строке заголовка информационного окна. Если этот аргумент пропустить, в строке заголовка появится надпись "Microsoft Excel", как на рис. 7.1.

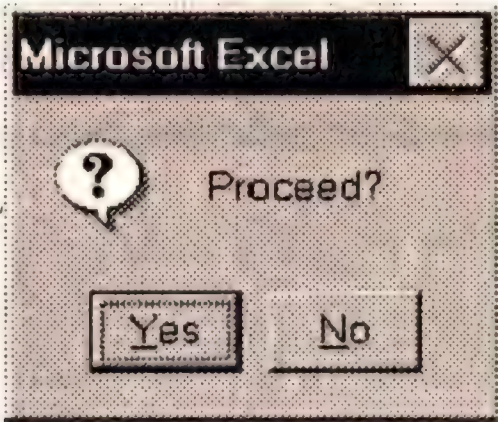


Рис. 7.1. Результат выполнения функции MsgBox

Функция MsgBox возвращает число, указывающее, какую именно кнопку нажал пользователь. Возможные возвращаемые значения можно найти в табл. 7.3. Кроме того, они объединены в перечисление VbMsgBoxResult.

Таблица 7.3. Возвращаемые значения функции MsgBox

Константа	Значение	Описание
vbOK	1	Была нажата кнопка OK
vbCancel	2	Была нажата кнопка Cancel
vbAbort	3	Была нажата кнопка Abort

продолжение ➤

Таблица 7.3 (продолжение)

Константа	Значение	Описание
vbRetry	4	Была нажата кнопка Retry
vbIgnore	5	Была нажата кнопка Ignore
vbYes	6	Была нажата кнопка Yes
vbNo	7	Была нажата кнопка No

Функция InputBox

Функция InputBox выводит на экран окно диалога с запросом на ввод данных пользователем. Чаще всего она описывается следующим образом:

```
InputBox(prompt [. title] [. default])
```

Переменная prompt задает сообщение, показываемое в окне диалога, а переменная title — сообщение в строке заголовка. Показываемое по умолчанию значение текстового поля определяется содержимым переменной default. Например, код `sName = InputBox("Enter your name.", "Name", "Albert")` приводит к появлению диалогового окна Name (Имя) с текстом Enter your name (Введите ваше имя), показанного на рис. 7.2.



Рис. 7.2. Результат действия функции InputBox

Функция InputBox возвращает строку, которую пользователь вводит в текстовое поле. В нашем примере эту строку будет содержать переменная sName.

Эту функцию можно использовать и для ввода чисел. Возвращаемую строку, например "12,55", можно преобразовать в число 12,55 с помощью функции Val, о которой мы поговорим чуть позже.

Функции для обработки строк

Вот набор функций, применяемых к строкам, содержащим как константы, так и переменные:

Функция Len

Функция Len возвращает длину строки, то есть количество входящих в нее символов. Соответственно, код `Len("January Invoice")` вернет значение 15.

Функции UCase и LCase

Функции UCase и LCase меняют регистр букв в строке на верхний или нижний соответственно. Код выглядит следующим образом:

```
UCase(string)
LCase(string)
```

Например, код

```
MsgBox UCase("Donna")
```

вернет строку DONNA.

Функции Left, Right и Mid

Функции Left, Right и Mid возвращают часть строки. В частности, код

```
Left(string, number)
```

возвращает число number символов с начала строки string. Код

```
Right(string, number)
```

возвращает число number символов с конца строки string. Например, код

```
MsgBox Right("Donna Smith", 5)
```

возвращает значение Smith.

Функция Mid описывается следующим образом:

```
Mid(string, start, length)
```

Она возвращает число знаков length строки string, начиная с позиции start. Например, код

```
Mid("Library.xls", 9, 3)
```

возвращает значение xls. Если пропустить параметр length, например

```
Mid("Library.xls", 9)
```

будет возвращен остаток строки, начиная с позиции start.

Функции InStr, InStrRev

Синтаксис очень полезной функции InStr выглядит следующим образом:

```
InStr(Start, StringToSearch, StringToFind)
```

Она находит вхождение одной текстовой строки (StringToSearch) в другую текстовую строку (StringToFind) и возвращает положение начала искомого текста относительно крайнего левого знака просматриваемого текста. Переменная Start указывает позицию знака, с которой следует начинать поиск. Если этот аргумент пропущен, он считается равным 1. Например, код

```
MsgBox InStr(1, "Donna Smith", "Smith")
```

возвращает значение 7, так как слово Smith начинается с седьмой позиции строки Donna Smith.

Функция InStrRev отличается только тем, что поиск ведется в обратном направлении строки StringToSearch.

Функция Replace

Функция Replace, отсутствовавшая в Excel 97, используется для замены одного набора знаков в строке другим. Например, код:

```
MsgBox Replace("the car is red", "red", "blue")
```

возвращает строку "the car is blue".

Функции Str и Val

Функция *Str* преобразует число в строку. Например, код:

```
Str(123)
```

возвращает строку 123. Функция *Val* имеет обратное назначение, то есть преобразовывает строку в число, с которым потом можно, к примеру, выполнять арифметические операции. Скажем, код

```
Val("4.5")
```

возвращает число 4,5, а код

```
Val("1234 Main street")
```

возвращает число 1234. Имейте в виду, что эта функция не распознает знак доллара (\$) и знак запятой. Соответственно, код

```
Val("$12")
```

вернет значение 0, а не 12.

Функции Trim, LTrim и RTrim

Функция *LTrim* удаляет первый символ строки. Соответственно, функция *RTrim* удаляет последний символ строки. А функция *Trim* удаляет первый и последний символы одновременно. К примеру, код

```
Trim(" extra ")
```

возвращает слово *extra*.

Функции String и Space

Функция *String* позволяет быстро создать строку, состоящую из одного символа, повторенного заданное число раз. Например, код

```
sText = String(25, "B")
```

помещает в переменную *sText* строку, содержащую 25 букв *B*. Функция *Space* возвращает строку, состоящую из заданного числа пробелов. То есть код

```
sText = Space(25)
```

присваивает переменной *sText* строку из 25 пробелов.

Оператор Like и функция StrCmp

Оператор *Like* крайне полезен при сравнении двух строк. Разумеется, для этого можно использовать знак равенства, и выражение

```
string1 = string2
```

будет иметь значение *True* при идентичности двух строк. Однако оператор *Like* позволяет производить сравнение с учетом регистра букв или проверять сходство с определенным образцом. Выражение:

```
string Like pattern
```

возвращает значение *True*, если строка *string* соответствует заданному образцу *pattern*, и значение *False* в противном случае. В действительности выражение также может вернуть результат *Null*. Тип сравнимой строки, который использует оператор *Like*, зависит от параметров оператора *Option Compare*. В данном случае существуют два варианта, размещаемые в разделе описания модуля (в том же месте, где и оператор *Option Explicit*):

```
Option Compare Binary
```

```
Option Compare Text
```


По умолчанию используется первый вариант. При этом сравнение символов строки осуществляется в соответствии с кодом ANSI, как показано ниже:

$$A < B < \dots < Z < a < b < \dots < z < A < \dots < \emptyset < a < \dots < \emptyset$$

Если же выбран вариант Option Compare Text, сравнение производится без учета регистра букв, в зависимости от локальных настроек вашего компьютера. Это следующий порядок сравнения символов:

$$A = a < A = a < B = b < \dots < Z = z < \emptyset = \emptyset$$

При этом последним в порядке сортировки текста является символ "]", код ANSI которого равен 91. Это полезно знать, если потребуется поместить последний пункт в алфавитном порядке, — просто заключите его в квадратные скобки.

Функции оператора Like позволяют использовать групповые символы, а также списки и диапазоны символов. Например, знак вопроса ? используется для обозначения произвольного символа. Знак * обозначает группу символов. Знак # заменяет любую цифру от 0 до 9. Для поиска совпадений с любым единичным символом в списке charlist используется обозначение [charlist]. Если же требуется найти совпадение с любым единичным символом вне списка charlist, воспользуйтесь обозначением [!charlist].

Более подробную информацию можно найти в справочных файлах редактора VBA.

Функция StrComp также используется для сравнения двух строк. Ее описание выглядит следующим образом:

StrComp(string1, string2, [. compare])

Возвращаемое ей значение показывает соотношение между строками string1 и string2. Это может быть значение больше, меньше или равно.

Другие функции и операторы

Из множества функций языка VBA мы остановимся на функциях Is, определяющих атрибуты переменной или объекта, функциях преобразования и функциях If и Switch, которые возвращают условный результат.

Функции Is

Язык VBA содержит ряд функций Is, возвращаемое значение которых указывает, выполняется ли определенное условие по отношению к аргументу. Вы уже сталкивались с функцией IsMissing в одном из предыдущих упражнений. Вот еще несколько функций из этого же семейства.

Функция IsDate

Эта функция указывает, можно ли преобразовать выражение в дату. Назначение этой функции для ячейки приводит к оценке ее содержимого. Если содержимое ячейки является датой, функция возвращает значение True. Например, код

IsDate(Range("F3"))

вернет значение True, если в ячейке F3 находится дата.

Функция IsEmpty

Функция IsEmpty указывает, была ли переменная-аргумент инициализирована и содержит ли выделенная ячейка какое-нибудь значение. Например, код

```
If IsEmpty(Range("A1")) Then...
```

проверяет, есть ли что-нибудь в ячейке A1 или же она пуста.

Функция IsNull

Функция IsNull проверяет, имеет ли переменная значение Null. Переменные с этим значением не содержат данных. Имейте в виду, что код

```
If var = Null Then
```

всегда будет возвращать значение False, так как большинство выражений, содержащих значение Null, автоматически возвращают в качестве результата False.

Если переменная var имеет значение Null, правильнее будет написать:

```
If IsNull(var) Then
```

Функция IsNumeric

Функция IsNumeric указывает, является ли аргумент числом. Например, если ячейка A1 содержит данные 123 (даже если они отформатированы как текст), то условие:

```
If IsNumeric(Range("A1")) Then
```

будет выполнено. С другой стороны, если в ячейке содержится текст 123 Main Street, будет возвращено значение False.

Функция Immediate If

Описание функции Immediate If имеет следующий синтаксис

```
IIf(Expression, TruePart, FalsePart)
```

Если параметр Expression имеет значение True, будет возвращено значение TruePart. В противном случае функция вернет значение FalsePart. Например, следующий код приводит к появлению информационного окна, в котором указывается на то, является ли пустой первая ячейка активного листа.

```
Dim rng As Range
Set rng = ActiveSheet.Rows(1)
MsgBox IIf(IsEmpty(ActiveSheet.Cells(1, 1))), _
    "Ячейка пуста", "Ячейка не пуста")
```

Важно понимать, что функция Immediate If всегда оценивает оба аргумента — TruePart и FalsePart — хотя и возвращает только один из них. При этом можно столкнуться с побочным эффектом. Например, рассмотрим код

```
x = 0
y = IIf(x = 0, x^2, 1/x)
```

Он вернет ошибку «деление на ноль», несмотря на то, что функция возвращает результат 1/x только в том случае, когда x не равен нулю. Но функция оценивает это выражение в обоих случаях, в том числе и когда x = 0.

Функция Switch

Синтаксис функции Switch выглядит следующим образом:

```
Switch(expr1, value1, expr2, value2, ... exprn, valuen)
```


где `exprn` и `valuen` — выражения. Для работы функции требуется всего одна пара «выражение–значение», но действие функции будет более наглядным, если имеются по крайней мере две такие пары.

Если выражение `exprn` имеет значение `True`, функция `Switch` вернет соответствующее значение `valuen`. Как и в случае с функцией `IIf`, происходит оценка всех выражений. Если ни одно из них не имеет значения `True`, функция возвращает значение `Null`. Для проверки в этом случае используется функция `IsNull`.

В листинге 7.1 по расширению файла определяется его тип: `Template`, `Workbook` или `Add-in`.

Листинг 7.1. Функция `Switch`

```
Sub ShowFileType(FileExt As String)
```

```
Dim FileType As Variant
```

```
FileType = Switch(FileExt = "xlt", "Template", _  
                  FileExt = "xls", "Workbook", _  
                  FileExt = "xla", "Addin")
```

```
· Показ результата  
If Not IsNull(FileType) Then  
    MsgBox FileType  
Else  
    MsgBox "Неопознанный тип"  
End If
```

```
End Sub
```

Этот код содержит один нюанс. Так как функция `Switch` может вернуть значение `Null`, мы не можем использовать для возвращаемого значения тип `String`, что, на первый взгляд, выглядит вполне естественно:

```
Dim FileType As String
```

```
FileType = Switch(FileExt = "xlt", "Template", _  
                  FileExt = "xls", "Workbook", _  
                  FileExt = "xla", "Addin")
```

Этот вариант кода вполне имеет право на жизнь, если переменная `FileType` имеет значение `xlt`, `xls` или `xla`. В противном случае будет получено сообщение об ошибке `Invalid use of Null`. Именно поэтому в описании переменная `FileType` относится к типу `Variant`, что дает возможность присваивать ей различные типы данных, в том числе и ключевое слово `Null`. Впрочем, проблемы можно избежать с помощью оператора `Select Case`, о котором мы поговорим в следующей главе.

Преобразования единиц измерения

Функция `InchesToPoints` преобразует количество дюймов в количество точек. В Excel часто приходится вводить значения в точках, в то время как большинство пользователей привыкло к величинам в дюймах. В одном дюйме содержится 72 точки.

Эта функция особо полезна при работе со свойствами положения, например `Top` или `Left`. Так, свойство `Top` объекта `Chart` определяет положение верхней части диаграммы. Это положение отмеряется в точках, начиная с первого ряда листа.

Соответственно, чтобы сделать его значение равным 0,25 дюйма, нужно написать:

```
ActiveChart.ChartObject.Top = InchesToPoints(.25)
```

Существует также обратная функция `PointsToInches`, без которой не обойтись, если функция возвращает значение в точках, а не в дюймах.

Оператор Веер

Оператор Веер подает сигнал через внутренний динамик компьютера. Он полезен, если требуется привлечь внимание пользователя. Однако результат зависит от аппаратного обеспечения вашего компьютера, и в некоторых случаях применение данного оператора не даст никакого результата. Поэтому желательно предупреждать пользователей о его наличии в коде программы. В общем случае для этого лучше всего использовать строку состояния Excel, что делается с помощью свойства `StatusBar` объекта `Application`.

Обработка ошибок

Различные типы ошибок обсуждались в главе 3, но там ничего не было сказано о способах исправления ошибок, возникающих во время выполнения программы. Для данной цели в языке VBA существует набор инструментов (`On Error`, `Resume`, объект `Err` и т. п.), и этой теме вполне можно посвятить целую главу книги.

Невозможно переоценить правильную работу над ошибками. Если вы собираетесь стать профессиональным разработчиком приложений, необходимо детально освоить соответствующие процедуры.

Разумеется, при написании программ для собственного использования рутинные процедуры по отладке кода не имеют большого значения. В конце концов, при обнаружении ошибки в программе приложение остановит ее выполнение, покажет окно с сообщением об ошибке и выделит подозрительный код. Это позволит внести необходимые изменения и тем самым решит проблему. Но вряд ли стоит ожидать, что другие пользователи будут заниматься отладкой вашей программы.

Кратко опишем основные моменты процесса исправления ошибок. Более подробную информацию можно найти в моей книге *Concepts of Object-Oriented Programming in Visual Basic* (издательство Springer-Verlag).

Оператор On Error

Оператор `On Error` указывает на то, что делать при возникновении ошибок периода исполнения. Наиболее часто он используется в следующей форме:

```
On Error GoTo label
```

где слово `label` обозначает метку. Например, рассмотрим следующий код:

```
Sub example()
```

```
On Error GoTo ERR_EXAMPLE  
MsgBox SelectionCells.Count
```



```
Exit Sub
```

```
ERR_EXAMPLE:
```

```
MsgBox Err.Description, vbCritical  
Exit Sub
```

```
End Sub
```

Эта процедура просто показывает число выделенных в данный момент ячеек. При выделении диапазона ячеек свойство `Cells` возвращает их набор, а свойство `Count` затем возвращает их число. Однако если в выделенный набор попали не только ячейки, но и, к примеру, рисунок, диаграмма или другой объект, свойство `Cells` не сможет их обработать. Для таких случаев в программу добавлена процедура `ERR_EXAMPLE`. Строка

```
On Error GoTo ERR_EXAMPLE
```

переводит выполнение кода на метку `ERR_EXAMPLE`. За этой меткой следует так называемый *код обработки ошибок* (*error handling code*). При обнаружении ошибки следующей выполняется функция `MsgBox`, которая приводит к отображению диалогового окна, показанного на рис. 7.3. Это окно содержит сообщение с описанием ошибки, полученное от объекта `Error`, о котором мы поговорим в следующем разделе.

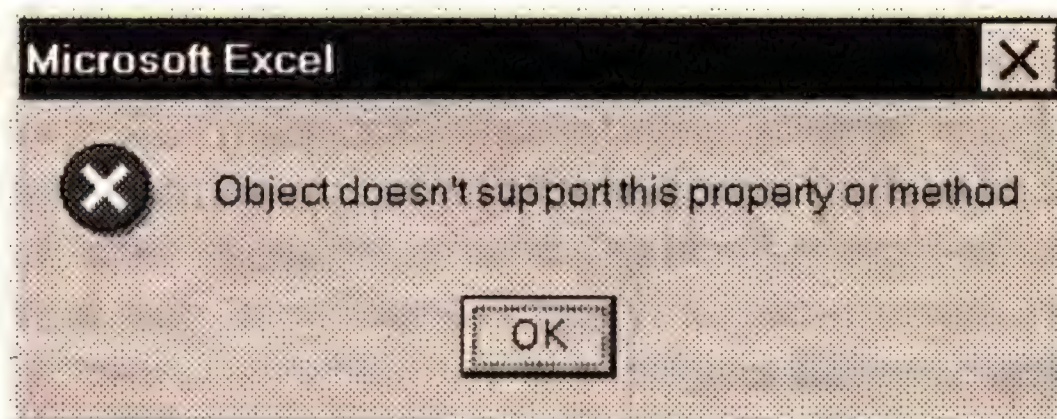


Рис. 7.3. Диалоговое окно с сообщением об ошибке

Обратите внимание на то, что перед меткой `ERR_EXAMPLE` расположена строка `Exit Sub`

Без этого оператора код обработки ошибок будет выполняться даже в том случае, если ошибки нет. Тем не менее пользователи часто пропускают эту строку. Учтите, что после метки всегда должно стоять двоеточие.

Объект Error

Объект `Error` принадлежит к объектной модели Excel. Его наиболее важными свойствами являются:

- `Number` — номер ошибки;
- `Source` — имя объекта, ставшего источником ошибки;
- `Description` — текстовая строка, описывающая ошибку.

Применение к этому объекту метода `Clear` возвращает все свойства объекта в исходное состояние. Например, свойство `Number` принимает значение 0, что эквивалентно отсутствию ошибок.

Оператор On Error GoTo 0

Оператор

On Error GoTo 0

отменяет действие любого расположенного ранее оператора On Error GoTo label. В результате любая возникающая впоследствии ошибка будет обработана своим собственным способом.

Оператор On Error Resume Next

Оператор

On Error Resume Next

приводит к выполнению кода, расположенного непосредственно за строкой, содержащей ошибку. Для данной формы оператора On Error существуют два варианта использования. Во-первых, с его помощью можно проигнорировать ошибку. Например, код:

```
Sub example()
```

```
On Error Resume Next
```

```
MsgBox Selection.Cells.Count
```

```
End Sub
```

сообщит о количестве выделенных ячеек, если действительно выделен некий диапазон ячеек листа. В противном случае никаких действий предпринято не будет.

Другим важным способом применения этого оператора является *контекстная проверка ошибок (in-line error checking)*, то есть немедленное исследование строки, которая могла стать причиной ошибки. Посмотрим, как в данном случае будет выглядеть вышеприведенный код:

```
Sub example()
```

```
On Error Resume Next
```

```
MsgBox Selection.Cells.Count
```

```
If Err.Number <> 0 Then
```

```
    MsgBox Err.Description, vbCritical
```

```
End If
```

```
End Sub
```

Оператор Resume

В код обработки ошибок можно включить также оператор Resume. Он передает выполнение строке, следующей за строкой, содержащей ошибку. Результат включения этого оператора в код из предыдущего раздела будет выглядеть следующим образом:

```
Sub example()
```

```
On Error GoTo ERR_EXAMPLE
```

```
MsgBox Selection.Cells.Count
```

```
Exit Sub
```

```
ERR_EXAMPLE:
```



```
MsgBox Err.Description, vbCritical  
Resume Next
```

```
End Sub
```

Оператор Resume имеет три разновидности:

- Resume
- Resume Next
- Resume ALabel

Оператор Resume приводит к повторному выполнению строки, ставшей причиной ошибки. Это полезно, если код обработки ошибок исправляет ситуацию и вы хотите проверить, насколько корректно теперь выполняется эта строка.

Проиллюстрируем действие этого оператора на примере. В листинге 7.2 код открывает книгу a:\text.xls. Но может оказаться, что этой книги не существует. Код обработки ошибок дает возможность ввести новое имя книги и снова выполнить операцию Open. Затем используется оператор Resume.

Листинг 7.2. Код обработки ошибок с оператором Resume

```
Sub test()
```

```
Dim sNew As String  
sNew = "a:\text.xls"  
On Error GoTo ERR_DISK  
Workbook.Open.sNew
```

```
Exit Sub
```

```
ERR_DISK:
```

```
If Err.Number = 1004 Then
```

```
    sNew = InputBox("Невозможно найти файл. Введите новый адрес или оставьте пустое  
    пространство для отмены операции")
```

```
    If sNew <> "" Then
```

```
        Resume
```

```
    Else
```

```
        Exit Sub
```

```
    End If
```

```
End If
```

```
End Sub
```

Третий вариант оператора — Resume ALabel — передает выполнение кода на строку с меткой ALabel.

8

Управляющие операторы

Рассмотрение языка VBA завершится рассказом об управляющих операторах, которые меняют процесс выполнения программы.

Оператор If...Then

Оператор If...Then изменяет порядок выполнения программы в зависимости от проверки некоторого условия. Вот его синтаксис:

```
If Условие Then
' Выполняемые операторы...
ElseIf Другое условие Then
' Другие операторы
Else
' Остальные операторы
End If
```

Можно включить более одной конструкции ElseIf, в то время как конструкция Else является необязательной. Также существует возможность записать этот код в виде одной строки, особенно при отсутствии конструкций ElseIf и Else. Вот пример кода, который удаляет выделенный набор активного листа, если он содержит более одной ячейки:

```
If Selection.Count > 1 Then Selection.Delete
```

Следующий код меняет цвет выделенного набора в соответствии с его положением. Выделенные ячейки в рядах с четными номерами будут окрашены в красный цвет, а в рядах с нечетными номерами — в синий.

```
Dim oCell As Range
For Each oCell In Selection.Cells
    If (oCell.Row Mod 2) = 1 Then
        ' Это четный ряд
        oCell.Interior.ColorIndex = 3 ' Окрашиваем в красный цвет
    Else
        oCell.Interior.ColorIndex = 5 ' Окрашиваем в синий цвет
    End If
Next
```


Цикл For...Next

Оператор For...Next используется для циклического повторения блока кода (то есть одной или нескольких линий). Для краткости этот цикл называют просто циклом For. Вот его основной синтаксис:

```
For counter = start To end
' Здесь находится блок кода
Next counter
```

При первом выполнении блока переменная counter принимает значение start, а при каждом последующем выполнении блока ее значение увеличивается на единицу. Цикл завершается, как только значение переменной counter превышает значение переменной end. То есть блок кода выполняется $end - start + 1$ раз, и каждый раз значение переменной counter модифицируется.

В принципе, в последней строчке можно не упоминать переменную counter, оставив только оператор Next. Это может ускорить выполнение цикла, но отрицательно сказывается на читаемости кода.

Для иллюстрации рассмотрим код, который циклически просматривает набор выделенных ячеек. Если ячейка содержит дату, шрифт окрашивается в красный цвет.

```
Dim i As Integer
Dim oCell As Range

For i = 1 To Selection.Count
' Просмотр следующей ячейки
Set oCell = Selection.Cells(i)
' Окрашивание шрифта, если в ячейке содержится дата
If IsDate(oCell) Then
    oCell.Font.ColorIndex = 3
End If
Next i
```

Цикл For часто используется для инициализации массива. Например, следующий цикл присваивает значение 0 каждому из 11 элементов массива iArray:

```
For i = 0 To 10
    iArray(i) = 0
Next i
```

Обратите внимание: переменная counter в некоторых случаях появляется внутри блока кода, как в вышеприведенном примере. В этом случае нужно быть особенно аккуратным, чтобы нечаянно не изменить ее значение, так как это приведет к путанице в цикле For. Увеличение этой переменной на единицу происходит автоматически при каждом выполнении цикла, и не стоит туда вмешиваться.

Выход из цикла For

Для преждевременного выхода из цикла For в языке VBA используется оператор Exit For. Например, код в листинге 8.1 ищет первую непустую ячейку в первом ряду активного листа. Если таковых не существует, появляется информационное окно с сообщением. Обратите внимание, что вопрос существования таких ячеек отслеживается с помощью переменной типа Boolean.

Листинг 8.1. Поиск первой непустой ячейки

```
Sub FindFirstNonEmpty()  
  
Dim oCell As Range  
Dim bNone As Boolean  
  
bNone = True  
For Each oCell In ActiveSheet.Rows(1).Cells  
    If Not IsEmpty(oCell) Then  
        oCell.Select  
        bNone = False  
        Exit For  
    End If  
Next  
  
If bNone Then MsgBox "В первом ряду все ячейки заполнены", vbInformation  
  
End Sub
```

С помощью ключевого слова **Step** можно контролировать величину и направление шага. Например, в следующем коде значение переменной *i* возрастает на два при каждом выполнении блока:

```
For i = 1 to 10 Step 2  
    ' Здесь идет блок кода  
Next i
```

В следующем цикле отсчет происходит от 10 до 1 с шагом -1. Это бывает полезно при исследовании набора (например, ячеек в ряду или столбце) снизу вверх.

```
For i = 10 to 1 Step -1  
    ' Здесь находится блок кода  
Next i
```

Цикл For Each

Цикл **For Each** является вариацией цикла **For** и предназначен для выполнения одной и той же группы действий над каждым объектом семейства или элементом массива. Его синтаксис выглядит следующим образом:

```
For ObjectVar in CollectionName  
    ' Здесь идет блок кода  
Next ObjectVar
```

Переменная **ObjectVar** имеет тот же тип, что и объекты из набора **CollectionName**. Блок кода выполняется один раз для каждого объекта в наборе.

Процедура **FindFirstNonEmpty** в листинге 8.1 иллюстрирует работу именно этого цикла.

Соответственно, при работе с набором объектов существует два варианта действий:

```
For Each Object in Collection  
    ' Здесь идет блок кода  
Next object
```

или

```
For i = 1 to Collection.Count  
    ' Здесь идет блок кода  
Next i
```


Помните, что при работе с наборами объектов Excel лучше использовать цикл For Each, так как в этом случае именно он работает быстрее.

Цикл Do

Цикл Do имеет несколько вариантов. Для их описания используем обозначение:

```
{While | Until}
```

которое представляет или ключевое слово While, или ключевое слово Until, но не оба этих слова сразу. В данном случае синтаксис использования данного цикла выглядит следующим образом:

```
Do {While | Until} condition
```

```
· Здесь находится блок кода
```

```
Loop
```

или

```
Do
```

```
· Здесь идет блок кода
```

```
Loop {While | Until} condition
```

В действительности есть и пятая возможность, так как можно убрать условие condition и записать:

```
Do
```

```
· Здесь находится блок кода
```

```
Loop
```

Различия между этими вариантами достаточно условны. Например, следующий код перебирает ячейки первого ряда активного листа, пока не обнаружит непустую ячейку:

```
i = 1
```

```
Do While IsEmpty(ActiveSheet.Rows(1).Cells(i))
```

```
    i = i + 1
```

```
Loop
```

```
ActiveSheet.Rows(1).Cells(i).Select
```

Если первый ряд активного листа не содержит пустых ячеек, с этим кодом возникнет проблема, но на данный момент не имеет смысла об этом беспокоиться. Рассмотрим еще один код сходного назначения:

```
i = 1
```

```
Do
```

```
    i = i + 1
```

```
Loop While Not IsEmpty(ActiveSheet.Rows(1).Cells(i))
```

```
ActiveSheet.Rows(1).Cells(i).Select
```

Разница между этими вариантами кода состоит в том, что в первом случае условие IsEmpty проверяется немедленно, перед выполнением кода внутри цикла Do. Соответственно, если первая ячейка пуста, условие не выполняется, в результате не будет выполнен код, расположенный внутри цикла Do. То есть эта ячейка будет выделена, как и требуется.

Во втором случае условие проверяется в конце цикла, поэтому расположенный внутри код выполняется в первый раз вне зависимости от того, пуста ячейка или нет.

Для преждевременного выхода из цикла Do используется оператор Exit Do.

Оператор Select Case

Как вы уже видели, оператор If...Then... используется для выполнения различных задач на основе некоторых условий. Альтернативной конструкцией, которая часто оказывается более читаемой, является оператор Select Case. Его синтаксис выглядит следующим образом:

```
Select Case testexpression
  Case value1
    ' Оператор, который выполняется, если testexpression = value1
  Case value2
    ' Оператор, который выполняется, если testexpression = value2
...
  Case Else
    ' Оператор, который выполняется в этом случае
End Select
```

Имейте в виду, что конструкцию Case Else можно опустить. Для иллюстрации работы этого оператора используем код из листинга 7.1, который фигурировал в предыдущей главе. Этот код определял тип файла по его расширению.

```
Sub ShowFileType(FileExt As String)

  Dim FileType As Variant

  Select Case FileExt
    Case "xlt"
      FileType = "Template"
    Case "xls"
      FileType = "Workbook"
    Case "xla", "utl"
      FileType = "Addin"
    Case Else
      FileType = "Неизвестный тип файла"
  End Select

  ' Отображение результата
  MsgBox FileType

End Sub
```

Уверен, что вы сочтете данный код более читаемым, чем в предыдущем случае. Обратите внимание на предпоследний оператор case:

```
Case "xla", "utl"
```

Язык VBA позволяет размещать несколько условий, разделенных запятыми. Это полезно в тех случаях, когда к одному и тому же результату приводят несколько условий.

Заключение

Существует множество операторов языка VBA, о которых мы не упомянули в настоящих главах. Справочное руководство по этому языку занимает около 300 стра-

ниц. Но мы постарались дать вам основную информацию, необходимую для начала программирования в Excel.

На самом деле для выполнения большинства задач вполне достаточно небольшой части функций языка VBA, и, скорее всего, чтобы разобраться в объектной модели Excel, вам придется потрудиться намного больше, чем при освоении этого языка.

Рассказ о языке VBA будет завершен перечислением тем для дальнейшего изучения, которое вы сможете осуществить с помощью справочной системы VBA.

Функции, связанные с файлами

В языке VBA существует большое число функций, связанных с файлом и папкой, которые используются для сохранения проекта. Они перечислены в табл. 8.1.

Таблица 8.1. Функции VBA, связанные с файлами и папками

Функция	Описание
Dir	Поиск файла с определенным именем
FileLen	Определение длины файла
FileTimeDate	Определение времени создания файла
FileCopy	Копирование файла
Kill	Удаление файла
Name	Переименование файла или папки
RmDir	Удаление папки
MkDir	Создание новой папки

Кроме перечисленных в табл. 8.1 функций иногда требуется создать новый текстовый файл для хранения данных. Для этой цели в языке VBA существует набор функций, имеющих в заголовке оператор Open. Синтаксис этого оператора выглядит следующим образом:

```
Open Маршрут доступа For Режим As [#]Номер файла
```

После открытия файла можно его читать или редактировать.

Функции, связанные со временем и датой

В табл. 8.2 перечислены функции языка VBA, использующиеся для управления временем и датой.

Таблица 8.2. Функции управления временем и датой

Функция	Описание
Date, Now, Time	Узнать текущее время и дату
DateAdd, DateDiff, DatePart	Произвести вычисление даты
DataSerial, DataValue	Вернуть дату
TimeSerial, TimeValue	Вернуть время
Data, Time	Настроить дату или время
Timer	Время процесса

Функции форматирования

В табл. 8.3 перечислены примеры функций, которые используются для форматирования строк, чисел и дат.

Таблица 8.3. Примеры функций форматирования

Функция	Возвращаемое значение
Format(Date, "Long Date")	Thursday, April 30, 1998
Format(Time, "Long Time")	5:03:47 PM
Format(Date, "mm/dd/yy hh:mm:ss AMPM")	04/03/98 12:00:00 AM
Format(1234.5, "\$##,##0.00")	\$1,234.50
Format("HELLO", "<")	"hello"

Часть III. Объектная модель Excel и приложения

- ☐ Глава 9. Объектные модели
- ☐ Глава 10. Приложения Excel
- ☐ Глава 11. События Excel
- ☐ Глава 12. Пользовательские меню и панели инструментов
- ☐ Глава 13. Встроенные диалоговые окна
- ☐ Глава 14. Нестандартные диалоговые окна
- ☐ Глава 15. Объектная модель Excel
- ☐ Глава 16. Объект Application
- ☐ Глава 17. Объект Workbook
- ☐ Глава 18. Объект Worksheet
- ☐ Глава 19. Объект Range
- ☐ Глава 20. Сводные таблицы
- ☐ Глава 21. Объект Chart
- ☐ Глава 22. Смарт-теги

9

Объектные модели

В данной главе вам предстоит познакомиться с объектными моделями и их управляющим синтаксисом.

Как уже упоминалось, язык VBA лежит в основе ряда важнейших приложений Windows, включая Microsoft Excel, Word, Access, PowerPoint, Visual Basic. В пакете Office 2000 в число этих приложений входит еще и Outlook. Приложения, использующие язык VBA, носят название хост-приложений. Каждое из них усовершенствовало язык VBA с помощью объектной модели (иногда и не одной). Эта модель используется для работы с объектами, присущими данному приложению.

Фирма Microsoft обеспечила прикладной программный комплекс и связанные с ним продукты более чем дюжиной различных объектных моделей. Сюда входят объектные модели, а именно: Excel, Word, Access, DAO (Data Access Object), Outlook, PowerPoint, Binder, Graph, Forms, VBA, VB, ASP (Active Server Pages) и т. п. Разумеется, эта книга посвящена исключительно объектной модели Excel, и большая часть ее занята описанием данной модели. Впрочем, в главе, посвященной настройкам меню и панелей инструментов в Excel, будет дана некоторая информация по объектной модели Office.

Объекты, свойства и методы

Каждый объект Excel располагает набором *свойств* (*properties*) и *методов* (*methods*). Примерами объектов в Excel являются книги, листы, диаграммы, шрифты и т. п. Сюда включаются 192 объекта, в том числе скрытые и устаревшие.

Свойства

Термин *свойство* в данном случае используется в том же контексте, что и в обычном русском языке. Это особенность, атрибут или характеристика объекта. Например, объект Worksheet имеет 55 свойств, в число которых входят Cells, Name, ProtectionMode и UserRange. Значение свойства может принадлежать к любому действительному типу данных, скажем, Integer, Single, String или даже к одному из объектных типов данных.

Если значение свойства принадлежит к типу Integer, оно называется целочисленным. Целочисленные свойства настолько распространены, что компания Mi-

Microsoft создала большое число встроенных перечислений (их число достигает 152, с 1266 индивидуальными константами), позволяющих присваивать этим значениям символьные имена. Например, свойство Calculation объекта Application может принимать любое значение из следующего перечисления:

```
Enum XlCalculation  
    xlCalculationManual = -4135  
    xlCalculationAutomatic = -4105  
    xlCalculationSemiAutomatic = 2  
End Enum
```

Значение свойства может являться объектом. Например, свойство ActiveChart объекта Workbook возвращает объект Chart, имеющий свой собственный набор свойств и методов.

Так как объект Chart можно получить из объекта Workbook, его называют *дочерним* (*child object*). Соответственно, объект Workbook будет являться *родительским* (*parent*) по отношению к объекту Chart. Чуть позже мы подробно поговорим об этом типе связи.

Методы

Методом (method) называется действие, которое может быть выполнено над объектом. Например, с объектом Worksheet связан метод Protect, который защищает лист от редактирования.

В терминах программирования свойства и методы объекта являются всего лишь встроенными функциями и подпрограммами. Важно подчеркнуть, что в некоторых случаях можно столкнуться с достаточно произвольной трактовкой данных терминов. Например, элемент Item иногда называется свойством, а иногда методом, в зависимости от рассматриваемого объекта. Иногда даже сотрудники компании Microsoft не могут точно определиться.

Свойства и методы имеют общее название: *члены объекта (members)*. Их не стоит путать с дочерними объектами.

Семейства объектов

При работе с объектной моделью Excel (впрочем, как и с любой другой объектной моделью) часто требуется обработать несколько объектов сразу. Например, каждая ячейка в выделенном наборе является отдельным объектом (объект Range), как и каждый столбец и строка в открытом листе. Для таких случаев был разработан специальный объект, называемый *семейством (collection object)*.

Обычно семейства состоят из объектов одного типа. Например, в объектной модели Excel существует семейство Rows, в которое входят все строки рассматриваемого листа. Для каждого открытого листа существует свое семейство Rows. Как вы убедитесь, свойства и методы объекта Collection сконструированы с учетом необходимости управления семейством.

В общем случае опознать семейство объектов можно по его названию, которое представляет собой множественное число от названия входящих в семейство объектов. Например, набор Worksheets содержит объекты Worksheet. Однако из этого правила существует ряд исключений. Например, набор Rows содержит

объекты Range. В объектной модели Excel не существует таких объектов, как Cell, Row или Column. Вместо них используется объект Range, которому посвящена глава 19 данной книги.

Семейства повсеместно встречаются в объектной модели Office. Более того, к ним принадлежит почти половина всех объектов объектной модели Excel. В табл. 9.1 перечислены наиболее часто используемые в Excel семейства.

Таблица 9.1. Некоторые семейства объектной модели Excel

Areas	FormatConditions	SeriesCollection
Axes	LegendEntries	Sheets
Borders	Names	Windows
ChartObjects	PivotFields	Workbooks
Charts	PivotTables	Worksheets
DataLabels	Points	
Filters	Range	

Еще раз подчеркнем, что семейство представляет собой всего лишь специальный тип объектов, удовлетворяющий следующим требованиям:

- Наличие свойства Count, возвращающего число объектов набора. Это свойство невозможно отредактировать вручную, оно автоматически обновляется редактором VBA.
- Наличие метода Add, который позволяет добавлять в семейство новые объекты.
- Наличие методов Remove, Close и Delete или им подобных, позволяющих удалять объекты из семейства.
- Наличие метода Item, который дает доступ к любому объекту семейства. Объект при этом обычно определяется по имени или по индексу.

Вышеперечисленные требования не являются строгими. Существуют как семейства, не обладающие каким-либо из указанных свойств, так и семейства с множеством дополнительных свойств и методов. Например, семейства Areas и Borders не обладают методом Add, то есть пользователь лишен возможности добавлять туда новые объекты.

В Excel существуют и очень сложные семейства, обладающие рядом свойств и методов, присущих исключительно типу объектов, который в них содержится. Например, с семейством Sheet связаны 10 свойств и 8 методов. Некоторые из них, например метод PrintOut, включены специально, чтобы дать возможность одновременно обрабатывать все входящие в семейство листы. В данном случае под термином *sheet* подразумевается как лист, содержащий обычные данные, так и лист с диаграммой.

Нумерация членов семейства

Нумерация членов семейства может начинаться как с нуля, так и с единицы. Большинство семейств в Excel и в VBA относится ко второму виду. Но некоторые более старые семейства имеют нумерацию, начинающуюся с нуля. Я думаю, что фирма Microsoft получила множество жалоб по поводу наличия таких семейств, и поэтому было решено поменять их нумерацию.

Перед обращением к членам семейства по номеру важно определить, к какому виду это семейство относится. В некоторых случаях необходимую информацию можно найти в справочной системе VBA, но проще воспользоваться образцом кода. Например, код:

```
For i = 1 To Selection.Cells.Count
    Debug.Print Selection.Cells(i).Value
Next i
```

совершенно корректен, так как нумерация семейства Cells начинается с единицы. Однако нумерация семейства UserForm, в которое входят все нестандартные диалоговые окна, загруженные в данный момент, начинается с нуля, поэтому код

```
For i = 1 To UserForms.Count
    Debug.Print UserForms(i).Value
Next i
```

приведет к появлению сообщения об ошибке. Правильный код выглядит так:

```
For i = 0 To UserForms.Count - 1
    Debug.Print UserForms(i).Value
Next i
```

Он сообщает о количестве загруженных в активный проект нестандартных диалоговых окон.

Иерархия объектной модели

Тот факт, что некоторые объекты Excel включены в другие объекты, имеет первостепенную важность и добавляет в объектную модель очень полезную структуру.

В процессе чтения различных источников создается впечатление, что не существует однозначного соответствия между родительскими и дочерними объектами. В нашем случае, если объект A имеет свойство или метод, который возвращает объект B, то объект A будет родительским по отношению к объекту B. Например, объект Range имеет свойство Font, возвращающее объект Font. Соответственно, объект Range является родительским по отношению к объекту Font. При этом последний является дочерним также по отношению к объекту ChartArea, с которым вы познакомитесь в главе 21. То есть число предков и потомков объекта неограниченно.

Не стоит понимать аналогию предок—потомок дословно. Помните, что иерархия объектов наполнена замкнутыми соотношениями предок—потомок. Например, объект Range является дочерним по отношению к объекту Worksheet, а объект Worksheet — по отношению к объекту Range. В большинстве моделей многие объекты имеют свойства, которые возвращают самый верхний объект в иерархии модели. В объектной модели Excel практически каждый объект обладает свойством Application, возвращающим объект Application, расположенный на верхнем уровне иерархии. Соответственно, практически каждый объект в данном случае можно считать родительским по отношению к объекту Application.

Иерархия объектов в объектной модели часто изображается в виде древовидной структуры, небольшая часть которой показана на рис. 9.1.

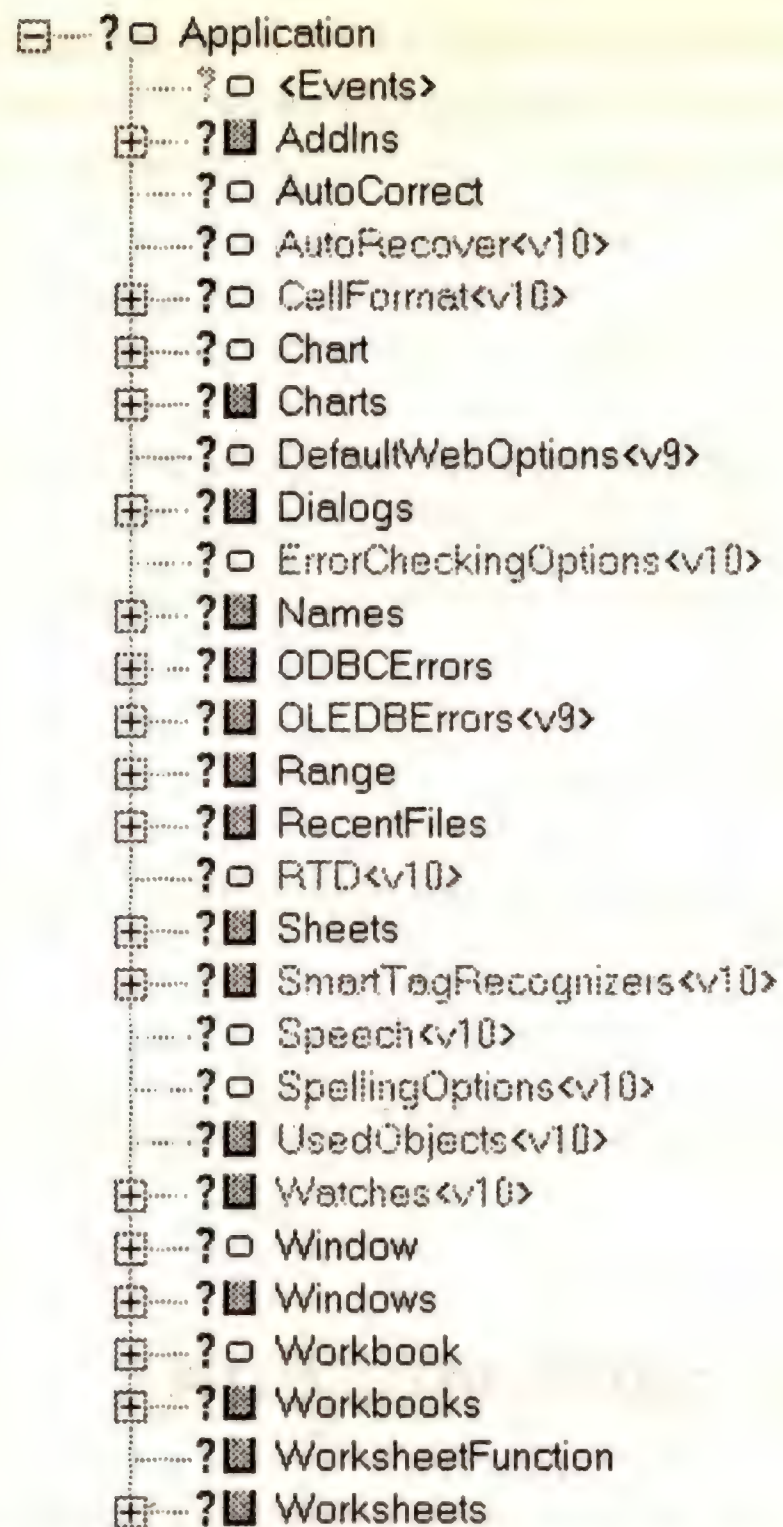


Рис. 9.1. Часть объектной модели Excel. Тег <vX> означает, что объект впервые появился в версии X приложения Excel

Синтаксис объектной модели

Пришло время обсудить базовый синтаксис, который используется при программировании объектной модели.

Ссылки на свойства и методы объекта осуществляются довольно просто. Если `objVar` — переменная объекта, а `AProperty` — свойство этого объекта, то доступ к чтению и редактированию данного свойства осуществляется следующим образом:

```
objVar.AProperty(требуемые параметры)
```

Например, следующий код задает шрифт в первом ряду активного листа:

```
* Описание переменной объекта
Dim rng As Range
```

```
* Сопоставление переменной rng первого ряда
Set rng = ActiveSheet.Rows(1)
```

```
* Указание имени шрифта
rng.Font.Name = "Arial"
```


Последняя строчка кода активизирует два свойства; свойство `Font` переменной `rng` возвращает объект `Font`, свойству `Name` которого присвоено значение `Arial`.

Метод `AMethod` для объекта `objVar` можно вызвать с помощью инструкции:

```
objVar.AMethod(требуемые параметры)
```

Обратите внимание, что в данном случае используется практически тот же самый синтаксис, что и при вызове обычных подпрограмм и функций. Просто теперь вместе с именем объекта требуется указать вызываемое свойство или метод. Например, можно применить метод `CheckSpelling` к ряду ячеек, обозначенному переменной `rng`:

```
rng.CheckSpelling
```

В качестве параметра данного метода используется, например, имя пользовательского словаря.

Объектные переменные

Как для доступа к свойству объекта, так и для вызова метода можно применять прямой и непрямой подход. В последнем случае используется объектная переменная.

Например, назначение свойства `Bold` объекту `Font` в первом ряду активного листа прямым методом выглядит следующим образом:

```
ActiveSheet.Rows(1).Font.Bold = True
```

В качестве альтернативы можно использовать объектную переменную. Вот два варианта ее применения:

```
Dim rng As Range  
Set rng = ActiveSheet.Rows(1)  
rng.Font.Bold = True
```

```
Dim fnt As Font  
Set fnt = ActiveSheet.Rows(1).Font  
fnt.Bold = True
```

Объектные переменные играют более важную роль, чем это может показаться на первый взгляд. Наиболее значимой причиной их использования является возможность увеличить читаемость кода при наличии нескольких ссылок на один и тот же объект. Например, вместо того, чтобы писать:

```
ActiveSheet.Rows(1).Font.Bold = True  
ActiveSheet.Rows(1).Font.Italic = True  
ActiveSheet.Rows(1).Font.Underline = False  
ActiveSheet.Rows(1).Font.Size = 12  
ActiveSheet.Rows(1).Font.Name = "Arial",
```

можно использовать объектную переменную `Font`. В результате код будет выглядеть следующим образом:

```
Dim fnt As Font  
Set fnt = ActiveSheet.Rows(1).Font  
fnt.Bold = True  
fnt.Italic = True  
fnt.Underline = False  
fnt.Size = 12  
fnt.Name = "Arial"
```


Оператор With

Оператор With в языке VBA позволяет существенно сократить обращение к объекту, что значительно уменьшает объем кода. Вот как будет выглядеть после применения этого оператора код из предыдущего раздела:

```
Dim fnt As Font
Set fnt = ActiveSheet.Rows(1).Font
With fnt
    .Bold = True
    .Italic = True
    .Underline = False
    .Size = 12
    .Name = "Arial"
End With
```

Общий синтаксис рассматриваемого оператора выглядит следующим образом:

```
With объект
    ' Здесь идут операторы
End With
```

Операторы в данном случае относятся к объекту, но при этом вам не требуется снова и снова использовать имя этого объекта.

Сокращение времени выполнения программы

Основным достоинством объектных переменных является не столько улучшение читаемости кода, сколько увеличение производительности. Например, для выполнения каждой из пяти строк в первой версии предыдущего кода требуется распознать ссылку на различные объекты Excel, а именно `ActiveSheet`, `Rows(1)` и `Font`. То есть приходится спускаться вниз по объектной модели, что занимает время.

Если же код использует объектную переменную типа `Font`, распознавание ссылки происходит только один раз и, соответственно, скорость выполнения возрастает. При наличии сотен или даже тысяч ссылок, которые требуется распознать, экономия времени может оказаться весьма значительной.

Ссылка на объект

Между объектными и неobjектными переменными существует большое различие. Как уже упоминалось, неobjектную переменную можно рассматривать как место в памяти компьютера, в котором хранятся некие данные. Рассмотрим, к примеру, код

```
Dim iVar As Integer
iVar = 123
```

Здесь переменная `iVar` представляет собой 4-байтовую ячейку в памяти, где хранится значение 123. В графическом виде это показано на рис. 9.2. В действительности значение 123 хранится в бинарном формате, но для данного обсуждения это несущественно.

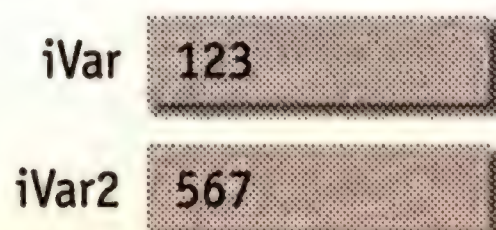


Рис. 9.2. Переменные типа Integer в памяти

Если далее написать:

```
Dim iVar2 As Integer  
iVar2 = iVar  
iVar2 = 567.
```

не стоит ожидать, что последняя строка кода окажет влияние на значение переменной `iVar`. Оно по-прежнему останется равным 123. Это связано с тем, что переменные `iVar` и `iVar2` представляют разные области памяти.

В свою очередь, объектная переменная является местом в памяти, где хранится адрес ссылки на объект, как показано на рис. 9.3. Именно поэтому объектные переменные иногда называют переменными-указателями. На рисунке объектная переменная `rng` указывает на объект типа `Range`, а именно на первый столбец активного листа.

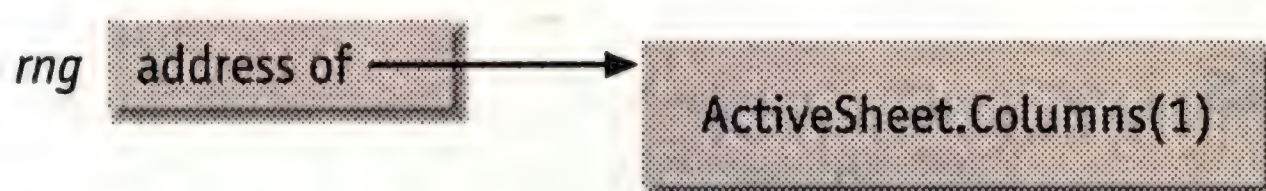


Рис. 9.3. Объектная переменная в памяти

Вот как выглядит код, соответствующий рис. 9.3:

```
Dim rng as Range  
Set rng = ActiveSheet.Columns(1)
```

Ссылка на один и тот же объект может осуществляться с разных переменных. Например:

```
Dim rng as Range  
Dim rng2 as Range  
Set rng = ActiveSheet.Columns(1)  
Set rng2 = rng
```

Ситуация, описанная этим кодом, показана на рис. 9.4.

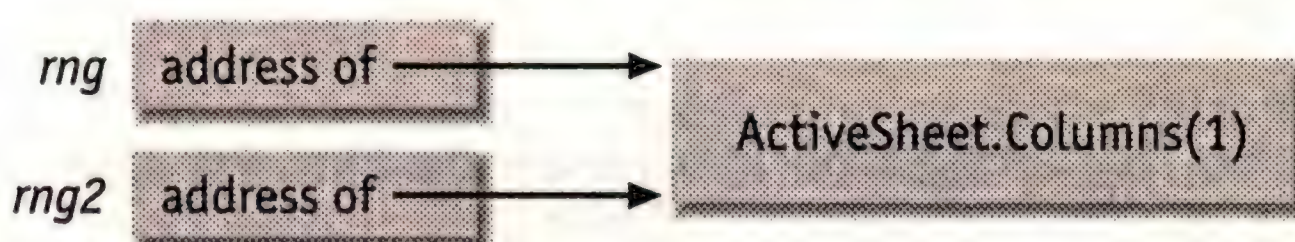


Рис. 9.4. Две объектные переменные указывают на один и тот же объект

Еще раз подчеркнем, что разные объектные переменные `rng` и `rng2` имеют одно и то же значение и, следовательно, указывают на один и тот же объект. То есть для редактирования первого столбца активного листа можно использовать любую из этих переменных.

В процессе программирования важно отслеживать все объектные переменные и объекты, на которые они указывают. Именно поэтому вышеописанная ситуация, когда на один объект указывают две переменные, должна возникать как можно реже и только при наличии веских причин. Очень легко забыть, что вид объекта был изменен с помощью первой переменной (например, `rng`), и, используя вторую переменную (`rng2`), думать, что вы работаете с исходным объектом.

Ключевое слово Nothing

Для очистки значения объектной переменной используется ключевое слово Nothing. Например:

```
Set rng2 = Nothing
```

Считается хорошим стилем программирования освобождать объектную переменную, когда в ней отпадает надобность, потому что это экономит ресурсы системы. Впрочем, объектной переменной это значение присваивается автоматически после завершения ее времени жизни.

Имейте в виду, что после освобождения переменная автоматически уничтожается, освобождая ресурсы. Однако перед этим должны быть уничтожены все ссылки на объект. Это еще одна причина не ставить в соответствие одному и тому же объекту более одной объектной переменной.

Оператор Is

Для сравнения значений двух обычных переменных достаточно написать:

```
If Var1 = Var2 Then...
```

Однако чтобы сравнить две объектные переменные и определить, не указывают ли они на один и тот же объект, используется другой синтаксис. Эта операция выполняется с помощью оператора Is:

```
If rng Is rng2 then...
```

Точно так же проверяется, свободна ли нужная вам переменная:

```
If rng Is Nothing Then...
```

Однако данный оператор в версиях VBA для Office 97 и Office 2000 не всегда работает корректно. Компания Microsoft осведомлена об этой проблеме. Например, код:

```
Dim Wks As Worksheet  
Dim Wks2 As Worksheet  
Set Wks = ActiveSheet  
Set Wks2 = ActiveSheet  
MsgBox Wks Is Wks2
```

покажет в качестве результата корректное значение True. В то время как аналогичный код:

```
Dim rng As Range  
Dim rng2 As Range  
Set rng = ActiveSheet.Rows(1)  
Set rng2 = ActiveSheet.Rows(1)  
MsgBox rng Is rng2
```

приведет к появлению неверного значения False. Чтобы исправить ситуацию, нужно записать предпоследнюю строку следующим образом:

```
Set rng2 = rng
```

В этом случае информационное окно покажет результат True.

Параметры объектной переменной по умолчанию

В большинстве объектных моделей многие переменные имеют заданный по умолчанию набор свойств и методов, который используется при отсутствии явного ука-

зания их на свойства и методы. Например, в объектной модели Microsoft Word переменная Range по умолчанию обладает свойством Text. Именно поэтому код:

```
Dim rng As Range
Set rng = ActiveDocument.Words(1)
rng = "Маша"
```

задает в качестве первого слова активного документа слово "Маша". Ведь приложение Word в последней строке кода сопоставляет с переменной заданное по умолчанию свойство, освобождая пользователя от необходимости писать:

```
rng.Text = "Маша"
```

К сожалению, ни в документации по Excel VBA, ни в объектной модели Excel нельзя найти упоминание о том, какие параметры присваиваются переменным по умолчанию. Поэтому я предлагаю не заострять внимание на этом вопросе в процессе программирования. Тем более что использование этих параметров уменьшает читаемость кода.

Исключением являются семейства объектов. Обычно им по умолчанию сопоставляется метод Item. Именно поэтому можно писать

```
Selection.Cells(4)
вместо более громоздкого
Selection.Cells.Item(4)
```

В данном случае использование заданного по умолчанию метода не создаст никакой путаницы, так что этим вполне можно пользоваться.

Глобальные параметры

Многие свойства и методы объекта Application можно использовать без указания имени этого объекта. Это так называемые глобальные параметры. Например, к ним относится свойство Selection, что позволяет писать:

```
Selection.Cells.Count
вместо
Application.Selection.Cells.Count
```

Для перечисления глобальных параметров существует специальный объект Global, параметры которого являются строгим подмножеством параметров объекта Application. Другими словами, не все свойства и методы объекта Application относятся к глобальным.

В табл. 9.2 перечислены глобальные параметры объектной модели Excel.

Таблица 9.2. Глобальные параметры объектной модели Excel

_Evaluate	CommandBars	Parent
_Run2	Creator	Range
ActiveCell	DDEAppReturnCode	Rows
ActiveChart	DDEExecute	Run
ActiveDialog	DDEInitiate	Selection
ActiveMenuBar	DDEPoke	SendKeys
ActivePrinter	DDERequest	Sheets
ActiveSheet	DDETerminate	ShortcutMenus

продолжение ↗

Таблица 9.2 (продолжение)

ActiveWindow	DialogSheets	ThisWorkbook
ActiveWorkbook	Evaluate	Toolbars
AddIns	Excel4IntMacroSheets	Union
Application	Excel4MacroSheets	Windows
Assistant	ExecuteExcel4Macro	Workbooks
Calculate	Intersect	WorksheetFunction
Cells	MenuBar	Worksheets
Charts	Modules	
Columns	Names	

10 Приложения Excel

Приложение Office можно определить как документ Office (например, база данных Access, книга Excel, документ или шаблон Word и презентация PowerPoint), содержащий специальные настройки. Эти настройки обычно представляют собой комбинацию процедур VBA и настроек меню и/или панелей инструментов для упрощения или автоматизации определенных задач. Сюда могут входить *служебные программы (utilities)*, выполняющие такие задачи, как, к примеру, распечатка текста или сортировка.

Подобное определение дает достаточную свободу действий. Например, добавив к шаблону Word еще один пункт меню, который просто вставляет подпись («Искренне ваш» и т. п.) в конец документа, мы можем рассматривать этот видоизмененный шаблон как приложение Word. Однако сомнительно, чтобы на подобное приложение нашелся покупатель.

Здесь мы хотим подчеркнуть, что приложение Office отличается от традиционных приложений Windows, таких как, например, Excel. Традиционные приложения Windows встроены в основной исполняемый файл. В случае с Excel это файл excel.exe. Разумеется, такое сложное приложение, как Excel, включает множество вспомогательных файлов. Сюда входят исполняемые файлы, справочные файлы, файлы библиотек объектных модулей, файлы ресурсов, информационные файлы, управляющие файлы ActiveX и вездесущие файлы DLL.

Приложения Office не имеют отношения к автономным выполняемым файлам. Они создаются внутри документов Office. В частности, приложения Access создаются внутри баз данных Access, приложения Excel — внутри книг Excel, приложения Word — внутри документов Word, а приложения PowerPoint — внутри презентаций PowerPoint. Приложения Office также могут быть созданы внутри шаблонов Office или встроенных программ.

В связи с этим возникает целый ряд вопросов по поводу распространения приложений Office. При их разработке требуется решить, куда поместить код приложения и какие средства предоставить пользователю для доступа к функциям этого приложения. Первый вопрос усложняется тем, что необходимо решить, следует ли давать пользователю доступ к коду приложения или нет.

Ответы на эти вопросы, как несложно догадаться, зависят от природы приложения.

Доступ к функциям приложений

Недавно я создал приложение Excel для известной сети ресторанов быстрого питания. Этой фирме требовалось оперативно отсылать данные о продажах в периферийные офисы в форме достаточно сложных сводных таблиц Excel. Специалисты по эксплуатации должны были фильтровать эти таблицы по различным значениям (формируя тем самым более мелкие сводные таблицы) и создавать различные диаграммы. Приложение имело и другие функции, но перечисленного вполне достаточно для иллюстрации сути происходящего.

В частности, основная сводная таблица содержала несколько типов данных (продажи, подсчет перечислений и т. п.) для ряда определенных областей продаж и типов магазинов. Одной из функций этого приложения являлась служебная программа для создания диаграмм. Но куда поместить этот код и как предоставить специалистам по эксплуатации доступ к формирующей диаграммы служебной программе?

Так как эта служебная программа напрямую включает в себя сводную таблицу, кажется разумным поместить кнопку с надписью Make Chart(s) (Создать диаграмму) непосредственно на лист с этой таблицей. Щелчок на этой кнопке приводит к появлению диалогового окна, показанного на рис. 10.1. Это окно позволяет пользователю выбирать различные типы данных и создавать на их основе одну или несколько диаграмм.

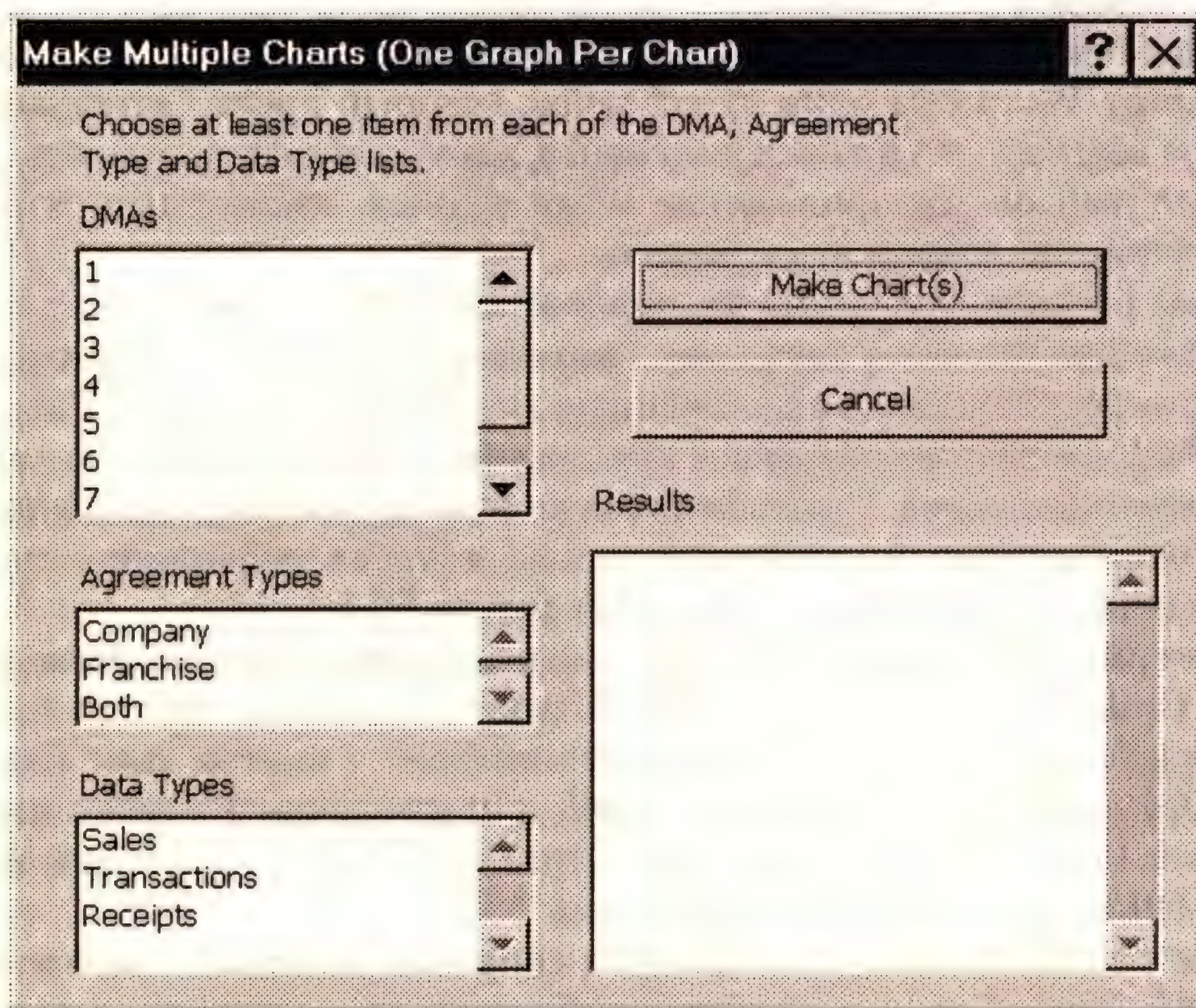


Рис. 10.1. Диалоговое окно служебной программы, предназначенной для создания диаграмм

Обеспечить доступ к формирующей диаграммы служебной программе, то есть к показанному на рис. 10.1 диалоговому окну, можно несколькими способами. Впрочем, эти способы подходят для доступа к любому макросу.

- Выбор нужного макроса в диалоговом окне Macro (Макрос), вызываемом командой Tools ► Macro ► Macros (Сервис ► Макрос ► Макросы). С этим окном вы познакомились в главе 4. Это наиболее эффективный способ для пользователей, которые пишут макросы и хотят иметь возможность их быстрого запуска. Но так как в этом окне показаны только названия макросов, вряд ли такой способ подходит для пользователей, которые не имеют понятия об их функциях. Этот способ также вряд ли следует рекомендовать для запуска часто используемых макросов.
- Если связать код служебной программы с событием Open, диалоговое окно будет появляться автоматически при открытии книги Excel. Подробный разговор о событиях пойдет в следующей главе.
- Поместить кнопку вызова диалогового окна непосредственно на лист.
- Поместить кнопку вызова диалогового окна на одну из существующих панелей инструментов Excel. Это можно сделать как программными методами, о которых мы поговорим в главе 12, так и с помощью пользовательского интерфейса. Последнее будет подробно объяснено чуть позже.
- Создать новую панель инструментов, на которую и будет помещена кнопка вызова диалогового окна. Это также можно сделать как программно, так и с помощью пользовательского интерфейса.
- Добавить новый пункт в одно из существующих меню Excel.
- Создать новое меню и добавить туда команду, вызывающую нужное диалоговое окно.

В моем случае требовалось, чтобы пользователь не мог вызвать служебную программу, формирующую диаграммы, если активный лист не содержит сводной таблицы. Поэтому был выбран вариант с размещением кнопки непосредственно на листе. Однако нельзя сказать, что остальные варианты в данном случае не будут работать.

С другой стороны, если рассматриваемое приложение имеет более широкую область применения, вероятно, имеет смысл создать кнопку на панели инструментов или добавить соответствующий пункт в меню. Я предпочитаю второй вариант, так как в этом случае служебную программу можно легко вызвать с помощью клавиатурных комбинаций.

Другими словами, решение о способе доступа к функциям приложения Office зависит от ряда вещей, включая сложность приложения, набор его функций и ваши собственные предпочтения.

Интерактивная работа с панелями инструментов и меню

Вне зависимости от того, где размещается командная кнопка для вызова макроса — на существовавшей ранее или специально созданной панели инструментов, — с помощью интерфейса Excel требуется указать, когда будет показываться эта панель инструментов. Для создания новых и для показа скрытых панелей инструментов используется вкладка Toolbars (Панели инструментов) диалогового окна Customize (Настройка), показанного на рис. 10.2. Его вызов осуществляется с помощью команды Customize (Настройка) меню Tools (Сервис).

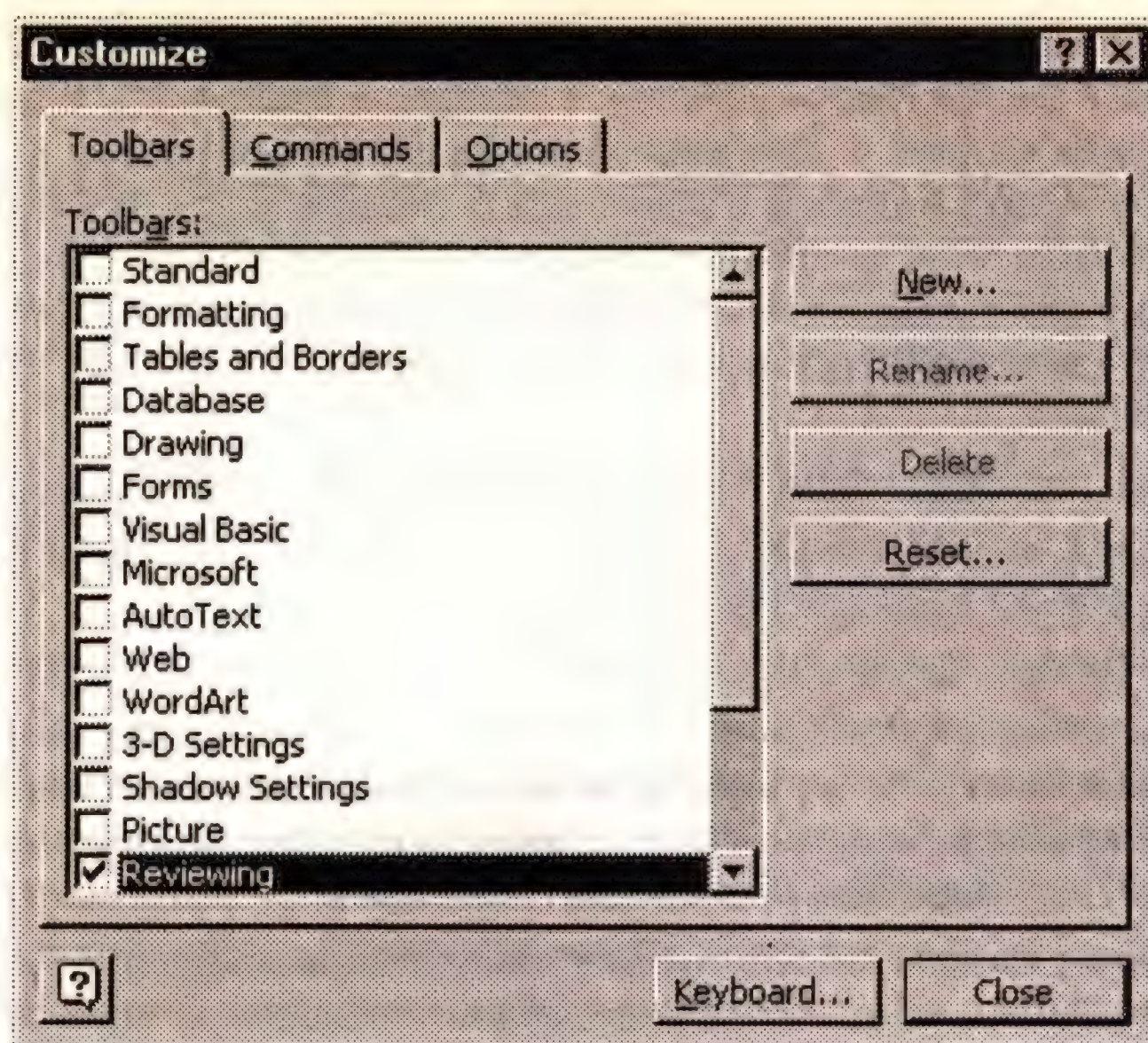


Рис. 10.2. Диалоговое окно Customize, открытое на вкладке Toolbars

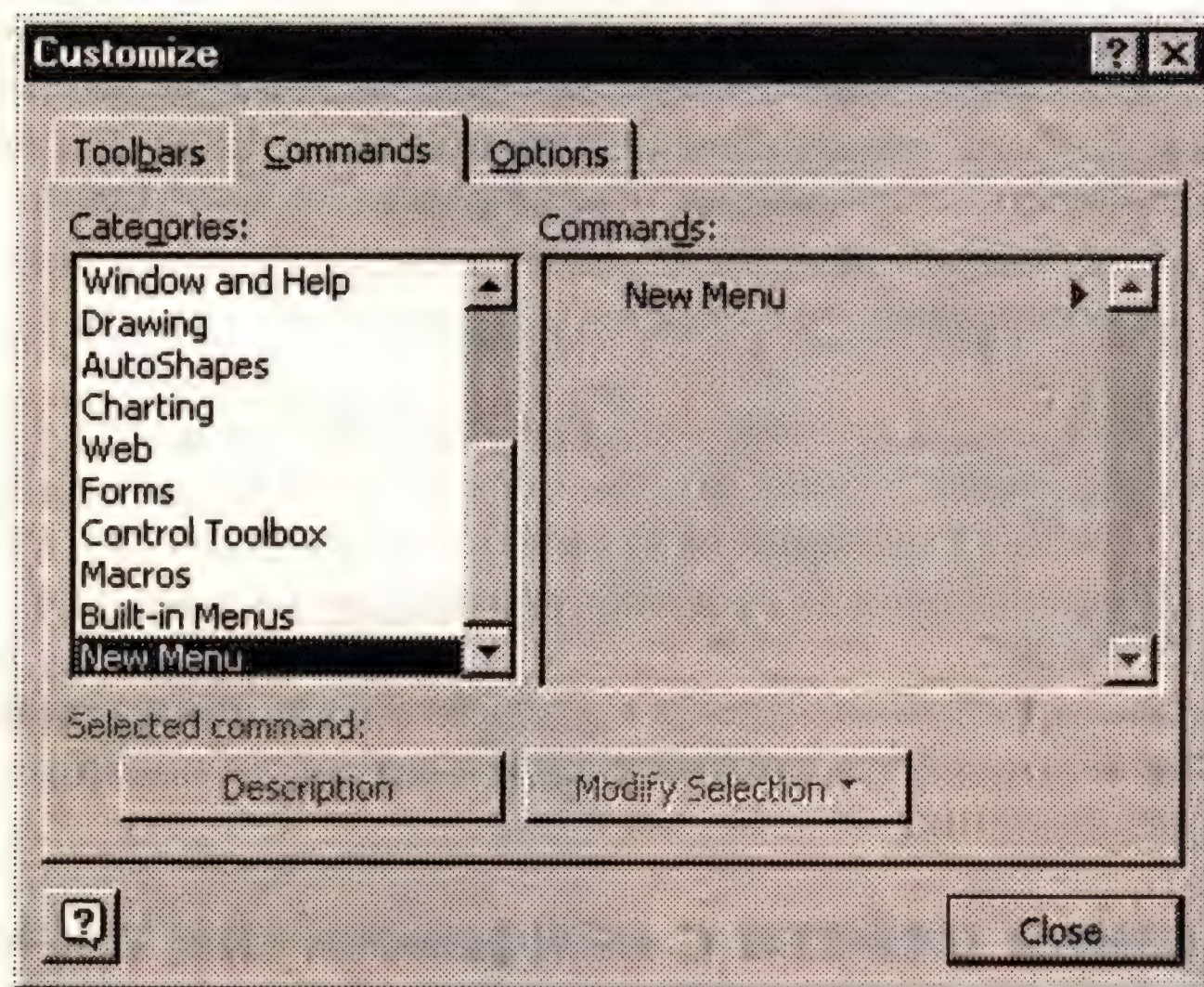


Рис. 10.3. Диалоговое окно Customize, открытое на вкладке Commands

Для создания новой панели инструментов щелкните на кнопке New (Создать). Появится диалоговое окно New Toolbar (Создание панели инструментов), в которое нужно ввести имя новой панели инструментов. После назначения уникального имени приложение Excel создаст новую панель инструментов и поместит ее имя в список Toolbars (Панели инструментов). Теперь можно заполнить эту панель кнопками.

Чтобы скрыть или сделать видимой любую панель инструментов, достаточно снять или установить флажок напротив названия нужной панели в списке Toolbars (Панели инструментов).

Также можно создать новое подменю, которое добавляется к существующему меню или панели инструментов. Эта операция выполняется на вкладке Commands (Команды) диалогового окна Customize (Настройка), как показано на рис. 10.3.

В списке Categories (Категории) нужно выбрать вариант New Menu (Новое меню) и перетащить пункт New Menu (Новое меню) на нужное меню или панель инструментов. Останется только щелкнуть правой кнопкой мыши и ввести имя нового меню в поле Name (Имя).

Назначение макросов меню и панелям инструментов

На вкладке Commands (Команды) диалогового окна Customize (Настройка) также можно назначать макросы существующим меню и панелям инструментов.

Хотя многие пользователи считают процесс настройки меню и панелей инструментов в Office чересчур сложным и запутанным, это ошибочное мнение. Достаточно помнить, что изначально меню и панели инструментов находятся в *режиме выполнения (run mode)*. То есть выбор пункта меню или нажатие кнопки на панели инструментов приводит к выполнению соответствующих действий. Но когда вы вызываете диалоговое окно Customize (Настройка), меню и панели инструментов переходят в *режим редактирования (edit mode)*. В этом режиме выбор пункта меню или нажатие кнопки панели инструментов производят совершенно другой эффект. В частности, щелчок правой кнопкой мыши на пункте меню приводит к появлению перечня его свойств. Кроме того, можно перемещать, удалять или добавлять пункты меню, просто двигая их мышью.

ВНИМАНИЕ

В режиме редактирования перенос пункта меню (или кнопки) из меню (или с панели инструментов) приводит к удалению этого элемента. Поэтому нужно быть очень внимательным, чтобы не сделать это непредумышленно.

Для создания кнопки на панели инструментов или пункта меню, которые приводят к выполнению определенного макроса, нужно выбрать вариант Macro (Макрос) в списке Categories (Категории) диалогового окна Customize (Настройка) и перетащить макрос из списка Commands (Команды) на нужное место в меню или на панели инструментов. Задача выполнена!

Необходимо упомянуть, что хотя описанный выше процесс подходит для разработчиков, вряд ли стоит заставлять клиента выполнять подобные действия. Альтернативой является программное создание нового элемента интерфейса, с которым вы познакомитесь в главе 12.

Место хранения приложений

В случае с приложением Excel для сети ресторанов быстрого питания все данные хранились в одной книге. Это было вполне допустимо, так как там не было дан-

ных, которые требовалось бы скрыть от пользователя. В результате книга стала совершенно самостоятельным объектом, для которого не требовалась процедура установки. Достаточно переслать нужную книгу из главного офиса в отделение по электронной почте. Однако даже в этом случае есть несколько вариантов действий.

- Хранить приложение и его данные непосредственно в документе, в котором они будут использоваться. Это подходит для самостоятельных приложений, подобных показанному на рис. 10.1, и для небольших макросов, которые повсеместно встречаются в данной книге в качестве примеров.
- Хранить приложение и его данные в шаблоне Excel. Это подходит, если шаблон будет служить основой для набора таблиц.
- Хранить приложение и его данные в скрытой книге Excel в загрузочной папке этой программы.
- Хранить приложение и его данные в надстройках Excel.

Каждый из этих вариантов имеет свои недостатки и преимущества, различные для разных приложений Office. Например, шаблоны намного чаще используются в Word, чем в Excel, а надстройки предпочтительней применять в Excel, а не в Access.

Загрузочная папка Excel

При загрузке Excel автоматически загружаются все крупноформатные таблицы, хранящиеся в загрузочной или в дублирующей ее папке. По умолчанию это подпапка основной папки Excel, называемойся *XlStart*. Изначально дублирующая папка отсутствует, но ее можно задать на вкладке *General* (Общие) диалогового окна *Options* (Параметры).

Так как содержимое этих папок открывается при запуске Excel в виде обычной книги, их макросы можно легко использовать в других книгах. Именно поэтому данные папки являются идеальным местом для сохранения макросов. Единственный недостаток состоит в том, что Excel в действительности открывает таблицы, хранящиеся в этих папках. Чтобы это предотвратить, нужно выбрать команду *Hide* (Скрыть) в меню *Window* (Окно) для активного листа.

Для доступа к макросам, хранящимся в данных папках, используется диалоговое окно *Macro* (Макрос). Затем с помощью пользовательского интерфейса Excel эти макросы можно назначить панели инструментов или меню. Впрочем, надстройки, о которых пойдет речь немного позднее, назначаются панелям инструментов или меню исключительно программным образом.

Книги, хранящиеся в любой из этих папок, можно использовать в качестве библиотеки макросов, доступной для любого из остальных листов. В них можно сохранять также созданные вами макросы, хотя надстройки, с которыми вы подробно познакомитесь чуть позднее, имеют большую гибкость и возможность настройки.

В этой книге предполагается, что вы собираетесь сохранять макросы в надстройках. Вскоре будет показано явное преимущество подобного подхода.

Шаблоны Excel

Шаблоны Excel являются отправной точкой для новых книг, листов, диаграмм и модулей кода. Создать шаблон очень просто. Достаточно открыть новую книгу и сохранить ее в формате *xlt* с помощью команды *Save As* (Сохранить как).

Предположим, что была открыта новая книга и в первую ячейку первого листа было введено число 123. Затем эта книга была сохранена в виде шаблона test.xlt. При следующем выборе команды New (Создать) в меню File (Файл) появится диалоговое окно, показанное на рис. 10.4. Останется выделить значок test.xlt и щелкнуть на кнопке OK, и приложение Excel откроет новую книгу и скопирует в нее данные из шаблона.

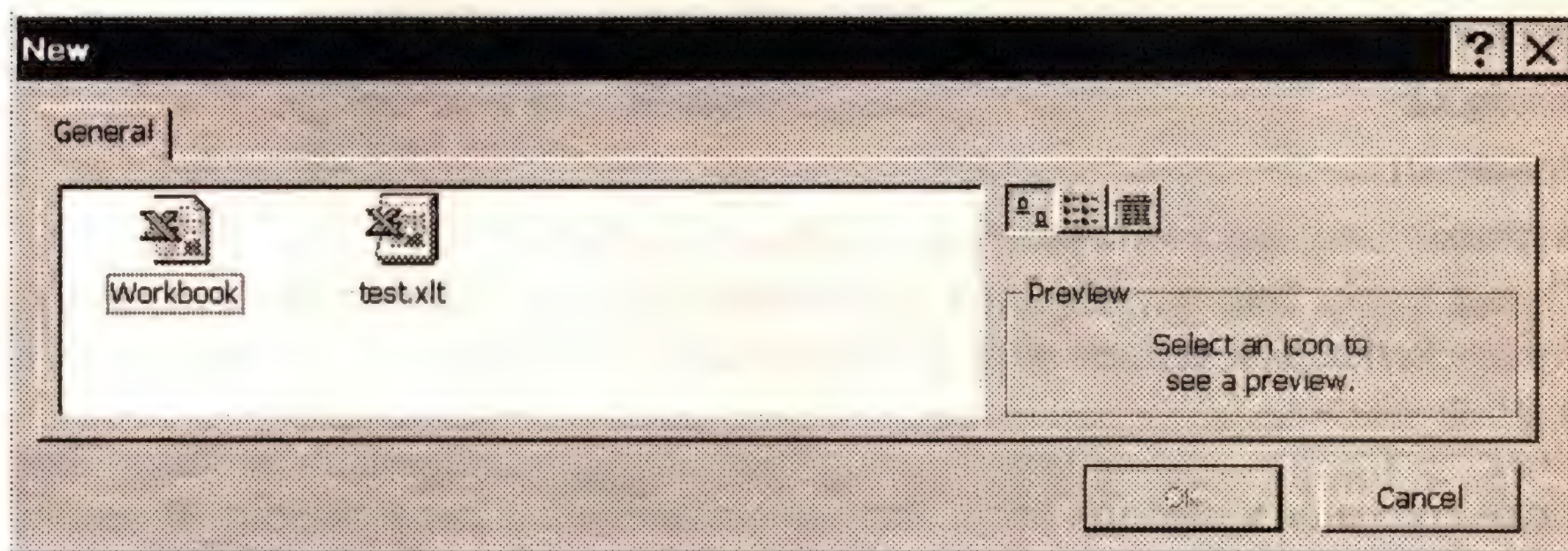


Рис. 10.4. Диалоговое окно New со значками шаблонов

Важно понимать, что как данные (в том числе и их форматирование), так и макросы копируются в новую книгу после того, как будут разорваны все связи между ней и шаблоном. Это коренным образом отличается от работы в приложении Word, в котором шаблоны связаны с документом. Изменения определенного характера, например добавление стилей или макросов, можно сохранить как в шаблоне, так и в самом документе, но Word никогда не копирует макросы из шаблона в документ. Кроме того, существует возможность открыть одновременно несколько шаблонов, каждый из которых будет каким-то образом влиять на вид документа. То есть шаблоны Word являются динамическими, в то время как шаблоны Excel — статическими.

Это значительно уменьшает полезность шаблонов Excel. Например, каждая книга, основанная на шаблоне, который содержит большое количество кода, будет снабжена копией данного кода. В некоторых случаях это является напрасной тратой места и сильно затрудняет управление кодом и его обновление. По этой причине я стараюсь, по возможности, не пользоваться шаблонами Excel.

Итак, перечислим элементы, которые передаются в новую книгу, открытую на основе шаблона:

- число и тип листов;
- форматирование ячеек и листа, созданное с помощью команд меню Format (Формат);
- стили ячеек;
- форматы страниц и настройки области печати для каждого листа;
- содержимое ячеек;
- графика, используемая на каждом из листов;
- пользовательские панели инструментов, макросы, гиперссылки, элементы управления ActiveX (пользовательские панели инструментов должны быть присоединены к шаблону);

- защищенные и скрытые области книги;
- параметры вычислений книги.

Приложение Excel также поддерживает несколько типов специальных шаблонов, называемых *автошаблонами* (*autotemplate*). Вот их имена:

- Book.xlt
- Sheet.xlt
- Chart.xlt
- Dialog.xlt
- Module.xlt
- Macro.xlt (для макросов четвертой версии Excel)

Если в папке XlStart находится шаблон Book.xlt, все книги, созданные при выборе значка Workbook в диалоговом окне, показанном на рис. 10.4, создаются на основе этого шаблона.

Если требуется, чтобы новая книга была отформатирована особым образом, достаточно создать шаблон Sheet.xlt и поместить его в папку XlStart. В результате при каждом вызове команды Insert ► Worksheet (Вставка ► Лист) Excel будет создавать копии всех листов шаблона Sheet.xlt и помещать их в текущую книгу. Имейте в виду, что будет создано именно то количество листов, которое содержится в шаблоне.

Остальные автошаблоны работают сходным образом.

Важно помнить, что все приложения Office используют для шаблонов одну и ту же заданную по умолчанию папку. То есть в этой папке могут одновременно храниться шаблоны для приложений Word, Excel, PowerPoint и Access. При этом единственным приложением, которое позволяет пользователю менять папку хранения шаблонов, является Word. Это можно сделать на вкладке File Locations (Расположение) диалогового окна, вызываемого командой Options (Параметры) из меню Tools (Сервис). Но имейте в виду, что указание нового адреса папки с шаблонами в Word приведет к изменению адреса этой папки для остальных приложений Office!

Надстройки Excel

Надстройки Excel представляют собой специальный тип книги, сохраняемый с расширением xla. Для связи надстроек с Excel требуется установить флажок напротив имени надстройки в диалоговом окне Add-Ins (Надстройки), показанном на рис. 10.5. Для его вызова используется команда Add-Ins (Надстройки) из меню Tools (Сервис).

Надстройку достаточно подсоединить один раз. Даже перезапуск приложения Excel не окажет влияния на ее связь с документом. Для разрыва этой связи требуется снять соответствующий флажок в диалоговом окне Add-Ins (Надстройки). Функции надстроек доступны во всех книгах Excel. То есть о них можно говорить как о расширении Excel.

Код надстроек приводит к появлению новых пунктов в меню или кнопок на панелях инструментов. Таким образом, пользователю предоставляется доступ к процедурам, выполняемым надстройкой. Этот код связан с событием Workbook_Open, благодаря чему меню (или панели инструментов) создаются/редактируются в момент подсоединения надстройки.

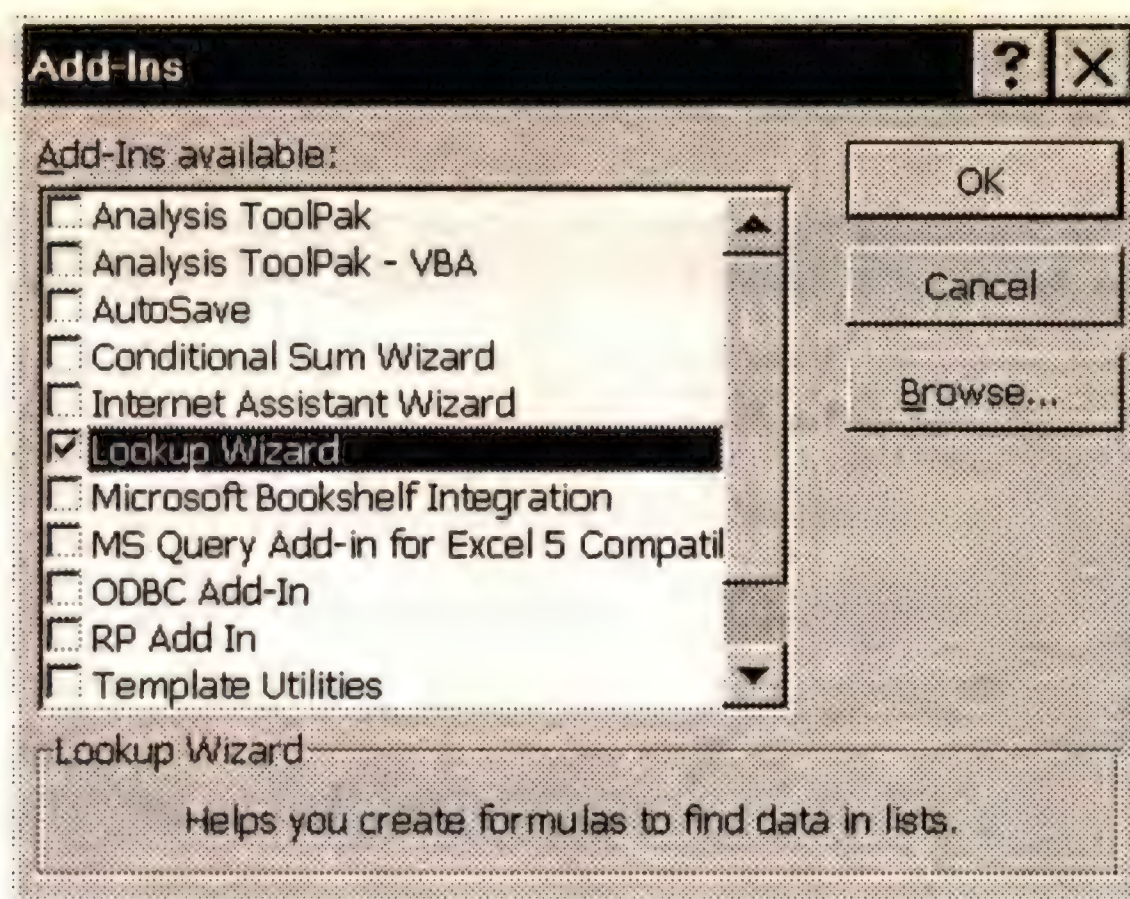


Рис. 10.5. Диалоговое окно Add-Ins

Создание надстроек

Создание надстройки представляет собой достаточно простой процесс. Начнем его с книги, которая называется SRXUtils.xls. (Это аббревиатура фразы Steven Roman's Excel Utilities — Служебные программы Excel Стивена Романа). Разумеется, данная книга содержит ряд макросов. Чтобы создать для нее надстройку, следуйте нижеприведенной инструкции.

1. **Компиляция проекта с помощью редактора VBA.** При редактировании и выполнении кода любой процедуры VBA сначала требуется произвести компиляцию этого кода, то есть его перевод на язык, понятный компьютеру. Эта операция автоматически выполняется приложением Excel при первом запуске кода. Впоследствии при выполнении кода компиляция не требуется, если, конечно, в код не было внесено никаких изменений. Для компиляции кода в книге SRXUtils.xls выберите команду **Compile** (Компилировать) в меню **Debug** (Отладка).
2. **Настройка свойств листа и проекта.** Выберите команду **Properties** (Свойства) в меню **File** (Файл) и откройте появившееся диалоговое окно на вкладке **Summary** (Документ), как показано на рис. 10.6. Информация из текстового поля **Title** (Название) будет фигурировать в диалоговом окне **Add-Ins** (Надстройки), показанном на рис. 10.7. Информация, введенная в поле **Comments** (Заметки), будет показана в нижней части диалогового окна **Add-Ins** (Надстройки). Следовательно, требуется заполнить оба этих поля. Затем свойства проекта задаются в редакторе VBA. В окне **Project** (Проект) выделите проект с именем SRXUtils.xls, а затем выберите команду **VBAProjectProperties** в меню **Tools** для вызова диалогового окна, показанного на рис. 10.8. Введите в соответствующие поля имя и описание проекта.
3. **Защита кода от просмотра.** Чтобы скрыть код от просмотра, воспользуйтесь диалоговым окном **Project Properties** (Свойства проекта). Перейдите на вкладку **Protection** (Защита), как показано на рис. 10.9. Установка флажка **Lock Project for Viewing** (Блокировать просмотр проекта) и ввод пароля защищает проект от просмотра и, соответственно, от записи. Такой проект отображается в окне **Project**

(Проект) редактора VBA, но чтобы раскрыть дерево иерархии, для этого проекта потребуется ввод пароля.

The screenshot shows the 'SRXUtils.xls Properties' dialog box with the 'General' tab selected. The dialog has a title bar with a question mark and a close button. Below the title bar are five tabs: 'General', 'Summary', 'Statistics', 'Contents', and 'Custom'. The 'General' tab contains several text input fields: 'Title' (containing 'SRXUtils'), 'Subject' (empty), 'Author' (containing 'sr'), 'Manager' (empty), 'Company' (containing 'rpi'), 'Category' (empty), 'Keywords' (empty), 'Comments' (containing 'Excel Utilities by SR'), 'Hyperlink base' (empty), and 'Template' (empty). At the bottom left is a checkbox labeled 'Save preview picture' which is unchecked. At the bottom right are 'OK' and 'Cancel' buttons.

Рис. 10.6. Ввод свойств надстройки в диалоговом окне Properties

The screenshot shows the 'Add-Ins' dialog box. It has a title bar with a question mark and a close button. Below the title bar is a section labeled 'Add-Ins available:' followed by a list box containing the following items: 'Analysis ToolPak', 'Analysis ToolPak - VBA', 'AutoSave', 'Conditional Sum Wizard', 'Internet Assistant Wizard', 'Lookup Wizard' (which is checked), 'Microsoft Bookshelf Integration', 'MS Query Add-in for Excel 5 Compatil', 'ODBC Add-In', 'SRXUtils' (which is highlighted), and 'Template Utilities'. To the right of the list box are three buttons: 'OK', 'Cancel', and 'Browse...'. At the bottom of the dialog, there is a text box containing 'SRXUtils' and another text box containing 'Excel Utilities by SR'.

Рис. 10.7. Диалоговое окно Add-Ins

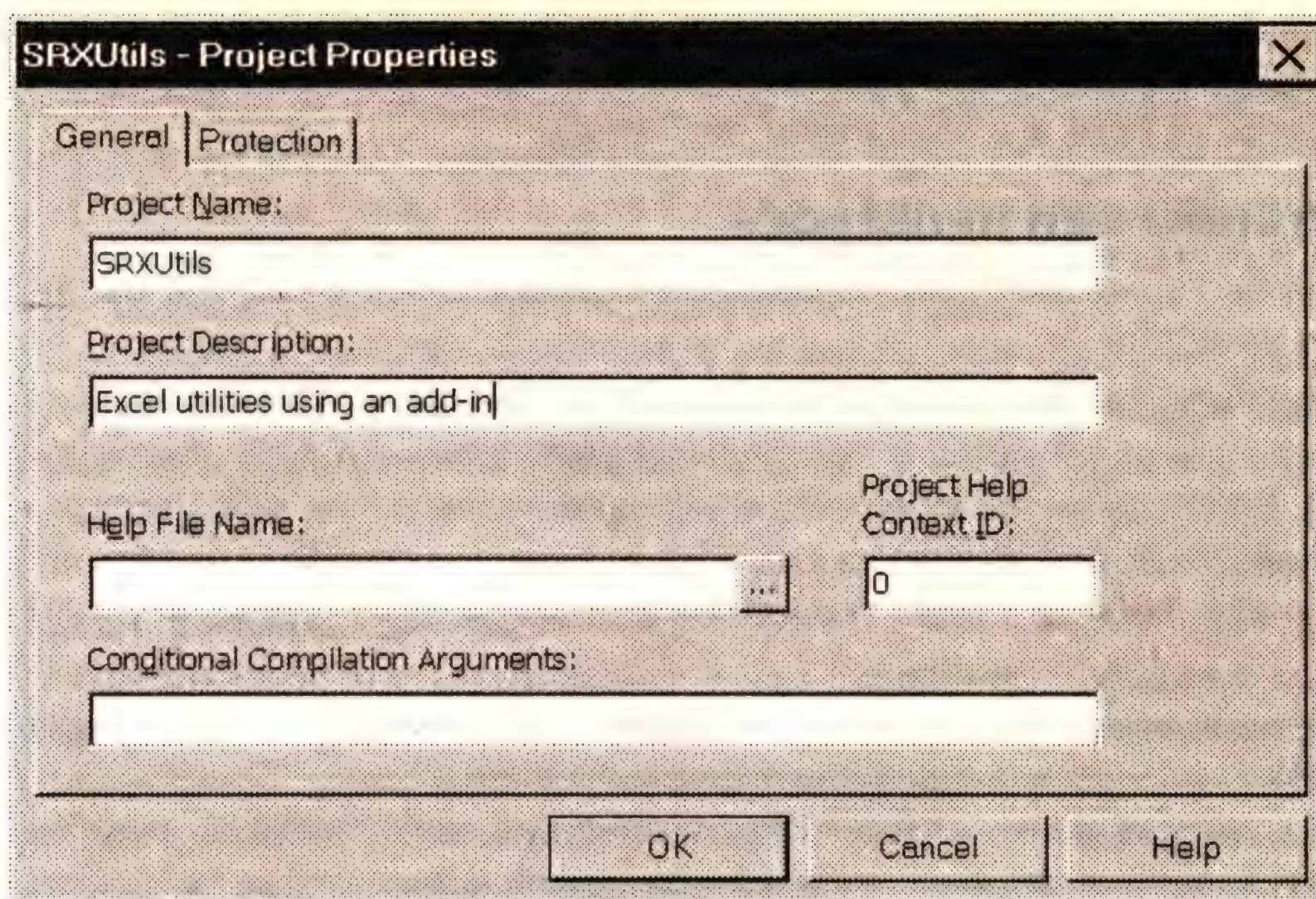


Рис. 10.8. Свойства проекта VBA

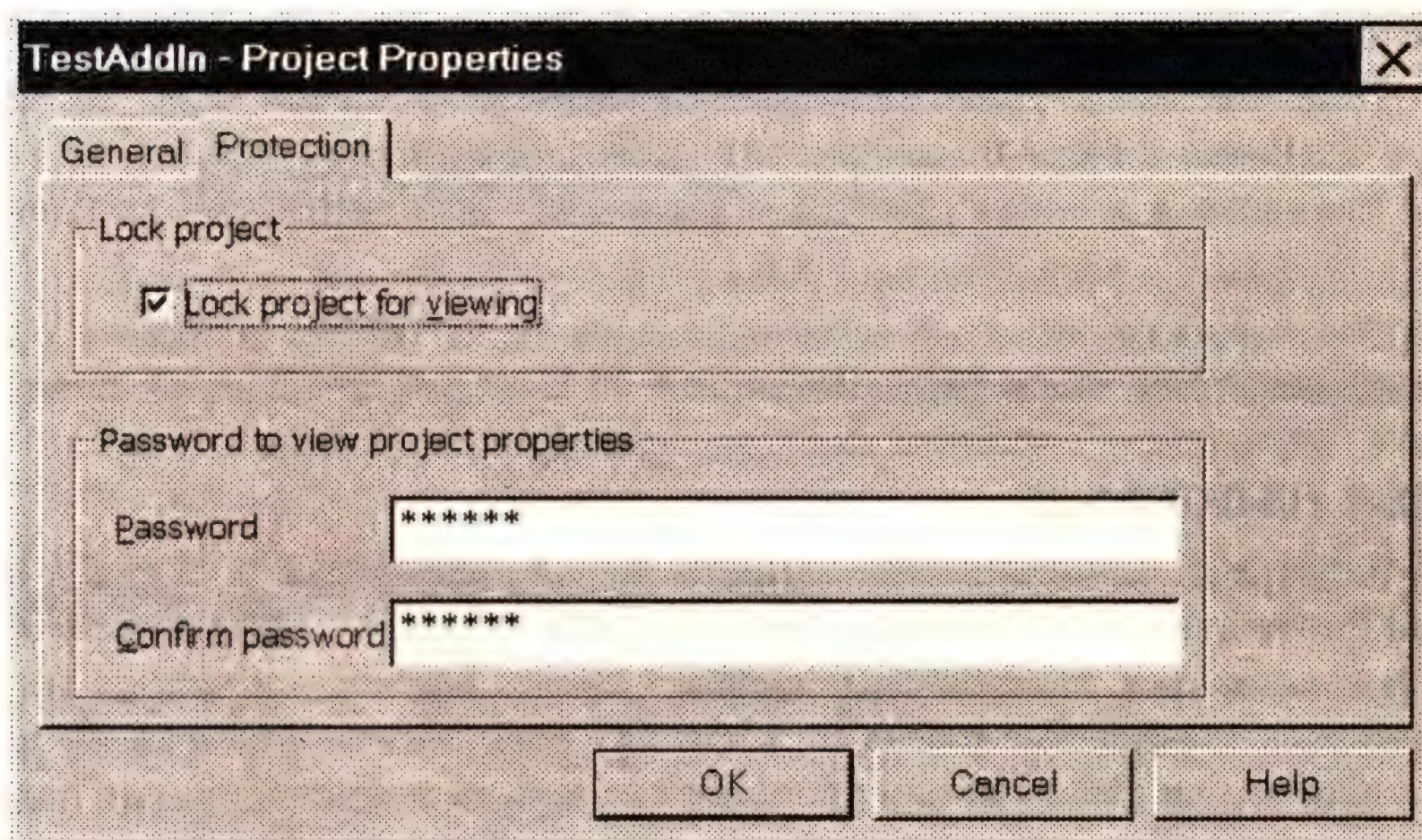


Рис. 10.9. Вкладка Protection диалогового окна Project Properties

4. **Сохранение книги как надстройки в выбранной вами папке.** Выберите команду Save As (Сохранить как) в меню File (Файл), а затем в раскрывающемся списке Save as Type (Тип файла) — вариант Microsoft Excel Add-In (*.xla) (Надстройка Microsoft Excel (*.xla)). Введите имя файла в поле File name (Имя файла) (в нашем примере это имя SRXUtils.xla) и щелкните на кнопке Save (Сохранить).

В каждой книге Excel есть свойство IsAddIn. Если это свойство имеет значение True, приложение считает, что книга снабжена надстройкой. Одним из следствий является невидимость книги. То есть невозможно просто задать свойство IsAddIn и сохранить проект в виде файла с расширением XLA, так как книга будет недоступна из пользовательского интерфейса Excel. К счастью, фирма Microsoft

организовала процесс сохранения книги в виде надстройки таким образом, что свойство `IsAddIn` автоматически примет значение `True`. О том, как вручную изменить это значение на `False`, мы поговорим чуть позже.

Характеристики надстроек

Перечислим характеристики надстроек, благодаря которым они отличаются от обычных книг.

- Книга и все листы надстройки являются скрытыми. Это необходимо, чтобы обезопасить надстройку от непредвиденных изменений со стороны пользователей. Если требуется включить в надстройку листы, их нужно сохранить в отдельной книге, в то время как код VBA сохраняется в файле надстройки. В этом случае при запуске Excel откроет оба файла.
- Как вы, вероятно, уже знаете, при попытке закрыть отредактированную книгу Excel появляется окно с вопросом, хотите ли вы сохранить сделанные изменения. Для надстроек не предусмотрено ничего подобного. То есть создатель надстройки может вносить в нее любые изменения, не беспокоясь о том, что пользователю будет показано окно с предупреждением, на которое тот вряд ли сможет адекватно отреагировать. Разумеется, создатель может в любое время сохранить результат редактирования надстройки с помощью команды `Save As` (Сохранить как).
- При открытии книги Excel запускается событие `Workbook_Open`. Если открывается обычная книга, пользователь может остановить данное событие, нажав и удерживая клавишу `Shift`. Событие `Open` для надстроек нельзя остановить.
- Названия макросов, которые используются в надстройках, невозможно увидеть в диалоговом окне `Macro` (Макрос).

Отладка надстроек

Отладка надстроек осуществляется таким же способом, как и отладка обычных книг. Вам не требуется снова ссылаться на исходный файл XLS. В частности, надстройки открываются, как обычные книги. Однако без знания пароля (предполагается, что надстройка защищена от доступа) невозможно увидеть ни кода надстройки, ни ее книги. Ввод пароля позволяет раскрыть дерево иерархии надстройки и получить доступ к расположенным ниже узлам. Чтобы увидеть код, достаточно выбрать узел проекта в окне `Project` (Проект). Если выбрать узел `ThisWorkbook` и открыть окно `Properties` (Свойства), появится возможность присвоить свойству `IsAddIn` значение `False`. Это сделает окно книги видимым и даст возможность его редактировать. После внесения необходимых изменений достаточно повторно откомпилировать код и вернуть свойству `IsAddIn` значение `True`.

ПРИМЕЧАНИЕ

Программа Excel 2000 поддерживает не только старую модель надстроек из Excel 97, но и более новую модель COM. Надстройки COM представляют собой ActiveX DLL или же выполняемый файл, который может быть связан с набором приложений Office. Эти надстройки создаются в редакторе Visual Basic или Visual C++, так как редактор VBA не позволяет создавать исполняемые файлы и файлы ActiveX DLL.

Удаление надстроек

Может быть, вы обратили внимание на то, что диалоговое окно Add-Ins (Надстройки), показанное на рис. 10.5, не снабжено кнопкой Delete (Удалить). Для удаления надстройки из списка требуется снять флажок, расположенный рядом с ее именем, переименовать файл XLA и снова установить флажок. Появится сообщение с вопросом, должен ли Excel удалить надстройку из списка.

Раз уж данный раздел посвящен свойствам надстроек, упомянем, что ввод нового имени в поле Title (Название) никак не отразится в диалоговом окне Add-Ins (Надстройки) до перезагрузки Excel.

Примеры надстроек

Начнем создание надстройки с оболочки. Это даст вам отправную точку для разработки законченных приложений Excel на основе надстроек путем добавления новых функций. Я строго рекомендую вам придерживаться описанного процесса.

В настоящей главе будет создана оболочка надстройки, которая всего лишь приводит к появлению информационного окна. На данный момент не стоит усложнять вопрос создания надстройки добавлением множества функций. С разработкой многофункциональных надстроек, а также с созданием для них пользовательских меню вы познакомитесь в главе 12.

Создание исходной книги

Для начала требуется создать новую книгу, которая послужит основой для надстройки. Я буду называть ее SRXUtils.xls, но вы можете использовать любое другое имя.

При редактировании надстройки изменения вносятся в лист SRXUtils.xls, который затем будет сохранен поверх текущей надстройки. Перед этим, разумеется, следует отключить текущую версию надстройки.

Настройка пользовательских меню

Для активации пробных функций оболочки требуется создать пользовательское меню. Подробно процесс создания меню и панелей инструментов будет объяснен в главе 12, а пока мы сведем его к минимуму, чтобы вы получили общее представление о том, как это происходит.

В данном случае пользовательское меню должно автоматически создаваться при загрузке надстройки и исчезать при ее отключении. Соответственно, код требуется связать с событиями Open и BeforeClose объекта ThisWorkbook, как показано в листинге 10.1.

Листинг 10.1. Программа обработки событий Open и BeforeClose

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
    DeleteCustomMenuItem
End Sub
```

```
Private Sub Workbook_Open
    CreateCustomMenuItem
End Sub
```


Данный код просто вызывает процедуры, создающие и удаляющие пользовательское меню. Эти процедуры нужно поместить в новый модуль кода, поэтому добавьте модуль к проекту SRXUtils и присвойте ему имя basMenus. Затем поместите в этот модуль процедуру CreateCustomMenuItem, описанную в листинге 10.2. На данный момент от вас не требуется полного понимания кода, так как он будет обсуждаться в главе 12. Помните, что данный код приводит к появлению пункта ActivateSheet в меню Custom, а выбор этого пункта меню запускает процедуру, описанную свойством OnAction. В данном случае это процедура ActivateSheet.

Листинг 10.2. Процедура CreateCustomMenuItem

```
Sub CreateCustomMenuItem()
Dim cbcpop As CommandBarControl
' Проверка наличия пользовательского меню
Set cbcpop = Application.CommandBars( _
    "Worksheet menu bar"). _
    FindControl(Type:=msoControlPopup, _
    Tag:="SRXUtilsCustomMenu")
If Not cbcpop Is Nothing Then Exit Sub

' Элемента управления не существует. Создадим его
Set cbcpop = Application.CommandBars( _
    "Worksheet menu bar"). _
    Controls.Add(Type:=msoControlPopup, _
    Temporary:=True)
Cbcpop.Caption = "Cu&stom"

' Задание свойства тега, чтобы потом найти его для удаления
With cbcpop.Controls.Add(Type:=msoControlButton, _
    Temporary:=True)
    .Caption = "&ActivateSheet"
    .OnAction = "ActivateSheet"
End With
End Sub
```

В модуль basMenus также следует поместить процедуру DeleteCustomMenuItem, описанную в листинге 10.3.

Листинг 10.3. Процедура DeleteCustomMenuItem

```
Sub DeleteCustomMenuItem()
Dim cbc As CommandBarControl
Set cbc = Application.CommandBars( _
    "Worksheet menu bar"). _
    FindControl(Type:=msoControlPopup, _
    Tag:="SRXUtilsCustomMenu")
If Not cbc Is Nothing Then cbc.Delete
End Sub
```

Выполнение функций надстройки

Так как служебная программа ActiveSheet (она активируется, когда пользователь выбирает в меню пункт ActivateSheet, созданный в листинге 10.2) очень проста, для нее не требуется собственного модуля кода. Достаточно добавить следующую процедуру в модуль basMain, который также следует создать:

```
Public Sub ActiveSheet()
    MsgBox "Это служебная программа ActiveSheet"
End Sub
```


Заключительные шаги

Напоследок требуется выполнить следующие шаги:

1. Скомпилируйте проект. Для этого используется меню **Debug** (Отладка).
2. Настройте свойства книги и проекта, как показано на рис. 10.6 и 10.8.
3. Установите защиту надстройки. На вкладке **Protection** (Защита) диалогового окна **Properties** (Свойства) установите флажок **Lock Project for Viewing** (Блокировать просмотр проекта) и введите пароль.
4. Сохраните надстройку под именем **SRXUtils.xla** в папке по вашему выбору.

Теперь все готово для испытания работы надстройки. Закройте книгу **SRXUtils.xls** и откройте новую книгу. Выберите в меню **Tools** (Сервис) команду **Add-Ins** (Надстройки) и щелкните на кнопке **Browse** (Обзор) в появившемся диалоговом окне. Найдите файл **SRXUtils.xla**, а затем установите флажок напротив его имени в списке **Add-Ins Available** (Доступные надстройки). В главном меню должен появиться пункт **Custom** с командой **ActiveSheet**. Выбор этой команды приводит к появлению информационного окна.

Как упоминалось ранее, по мере выполнения упражнений данной книги назначение этой надстройки станет понятнее.

11 События Excel

В процессе работы с Excel происходят различные *события*. Примером события может служить создание нового листа или изменение размера диаграммы. В Microsoft Excel определены 63 события.

При возникновении события запускается код для его обработки. Как вы вскоре убедитесь, этот код помещается в специальную процедуру. Он называется *кодом события (event code)*. Несложный код был написан в прошлой главе для событий Open и BeforeClose при создании надстройки SRXUtils.

События Excel естественным образом делятся на пять групп, перечисленных в табл. 11.1–11.5. Эти группы частично отражают уровень, на котором происходит событие, — уровень приложения (высший), уровень книги (средний) или уровень листа/диаграммы (низкий).

Для иллюстрации перечислим события, которые наступают при активации листа пользователем или при вызове метода Activate.

- Событие Activate. Оно наступает при каждой активизации листа.
- Событие SheetActivate на уровне книги, наступающее при каждой активизации листа в книге.
- Событие SheetActivate на уровне приложения. Оно наступает при активизации любого листа любой активной на данный момент книги. Однако вскоре вы убедитесь в том, что для запуска этого события требуется написать специальный код.

Свойство EnableEvents

Важно понимать, что ни одно событие Excel не наступит, если свойству EnableEvents не присвоено значение True. Это дает возможность программисту контролировать наступление событий. Свойство EnableEvent присуще объекту Application, то есть, к примеру, для предотвращения события Save при сохранении активной книги следует воспользоваться следующим кодом:

```
Application.EnableEvent = False  
ActiveWorkbook.Save  
Application.EnableEvents = True
```


События и объектная модель Excel

Объектная модель Excel включает в себя несколько объектов, которые существуют просто для удобства, позволяя включить события Excel в объектную модель. В код эти объекты не включаются. Вот их названия: AppEvents, DocEvents, ChartEvents, WorkbookEvents, OLEObjectEvents и RefreshEvents. Например, события, связанные с листом Excel, являются методами объекта DocEvents, который, в свою очередь, является дочерним по отношению к объектам Worksheet и Chart.

Доступ к процедуре обработки события

Вы, вероятно, хотите узнать, как написать процедуру обработки события. Коротко говоря, в Excel для каждого события существует специальная оболочка, в которую и помещается код процедуры.

Для иллюстрации рассмотрим событие SelectionChange для объекта Worksheet. На рис. 11.1 показано окно кода для объекта Sheet1 (Лист 1). Обратите внимание на то, что в раскрывающемся списке Object (Объект) выбран объект Worksheet. Это приводит к появлению в списке Procedure (Процедуры) перечня событий для данного объекта. Теперь остается лишь выбрать нужное событие.

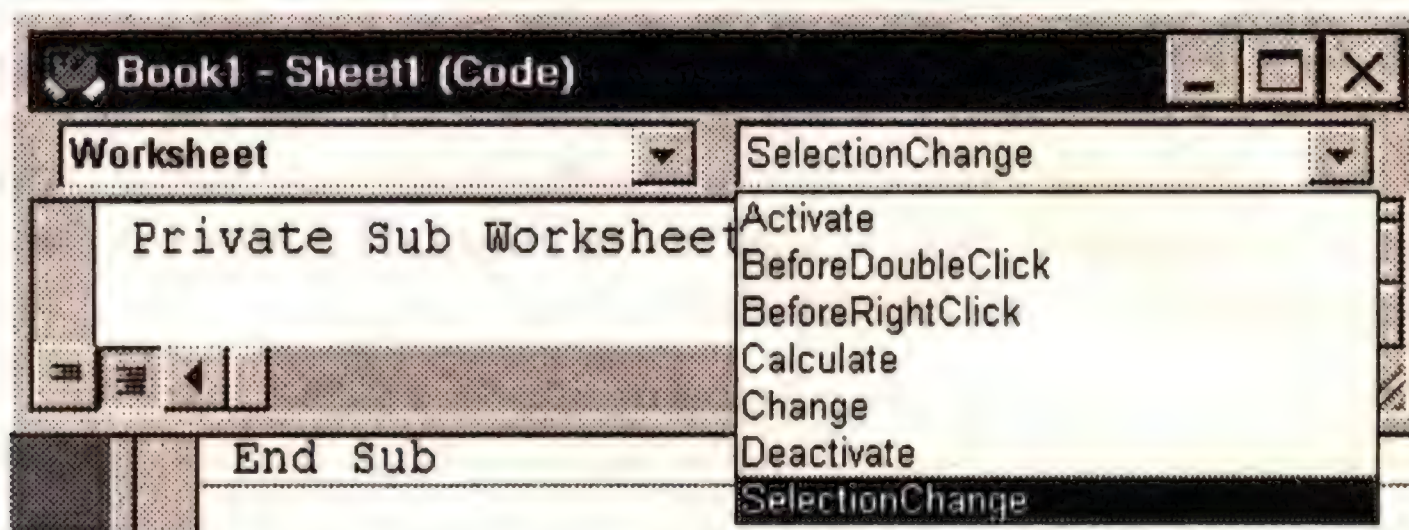


Рис. 11.1. События для объекта Worksheet

Выбор события SelectionChange влечет за собой автоматическое появление следующей оболочки кода:

```
Private Sub Worksheet_SelectionChange(ByVal Target As Excel.Range)
```

```
End Sub
```

При этом курсор окажется между этими двумя строками, приглашая вас начать ввод кода данного события.

Это событие запускается при изменении выделенной на листе области. Имейте в виду, что в качестве параметра Target будет фигурировать объект типа Range, представляющий новую выделенную область. Соответственно, код будет иметь доступ исключительно к новому набору выделенных объектов, что дает приложению Excel способ передать нам информацию, относящуюся к данному событию.

Описанный подход будет работать и для событий, связанных с книгой или диаграммой, в то время как события уровня приложения требуют несколько другой обработки. Но об этом мы поговорим чуть позже.

События объекта Worksheet

Процедуры для обработки событий объекта Worksheet содержатся в программном модуле, связанном с рабочей книгой. События, поддерживаемые этим объектом, перечислены в табл. 11.1.

Таблица 11.1. События объекта Worksheet

Событие	Описание
Activate	Запускается при активизации листа
BeforeDoubleClick	Запускается после двойного щелчка пользователя, но до стандартной реакции Excel
BeforeRightClick	Запускается после того, как пользователь щелкнет правой кнопкой мыши, но до наступления стандартной реакции Excel
Calculate	Запускается после пересчета листа
Change	Запускается при изменении данных листа
Deactivate	Запускается при переходе с текущего листа на другой
PivotTableUpdate (Только для Excel 10)	Запускается после обновления расположенной на листе сводной таблицы
SelectionChange	Запускается при изменении выделенной области листа

События объекта Workbook

Процедуры для обработки событий объекта Workbook находятся в программном модуле объекта ThisWorkbook, доступном в окне Project (Проект). События, поддерживаемые данным объектом, перечислены в табл. 11.2.

Таблица 11.2. События объекта Workbook

События	Описание
Activate	Запускается при активизации книги
AddinInstall	Запускается при установке рабочей книги в качестве надстройки
AddinUninstall	Запускается при удалении рабочей книги, которая служила надстройкой
BeforeClose	Запускается перед закрытием рабочей книги
BeforePrint	Запускается перед печатью рабочей книги
BeforeSave	Запускается перед сохранением рабочей книги
Deactivate	Запускается при переходе от текущей рабочей книги к другой
NewSheet	Запускается при вставке в книгу нового листа
Open	Запускается при открытии рабочей книги
PivotTableCloseConnection (Только для Excel 10)	Запускается при разрыве связи сводной таблицы с источником данных
PivotTableOpenConnection (Только для Excel 10)	Запускается при установлении связи сводной таблицы с источником данных
SheetActivate	Запускается при активизации любого листа, входящего в книгу
SheetBeforeDoubleClick	Запускается после двойного щелчка пользователя на любой из листов, входящих в книгу, но до стандартной реакции Excel

События	Описание
SheetBeforeRightClick	Запускается после того, как пользователь щелкнет правой кнопкой мыши на любом из листов, входящих в книгу, но до стандартной реакции Excel
SheetCalculate	Запускается после пересчета любого листа, входящего в книгу, или после отражения измененных данных на диаграмме
SheetChange	Запускается при изменении содержимого любых ячеек
SheetDeactivate	Запускается при переходе с одного листа на другой
SheetSelectionChange	Запускается при изменении выделенной области на любом листе. Это не относится к листам с диаграммами
WindowActivate	Запускается при активизации окна, если в одной книге открыто несколько окон
WindowDeactivate	Запускается при переходе от текущего окна к другому, если в одной книге открыто несколько окон
WindowResize	Запускается при изменении размеров окна

Имейте в виду, что, удерживая клавишу Shift при открытии книги, можно отметить событие Open для объекта Workbook.

События объекта Chart

В данном случае события делятся на связанные с диаграммой в целом и связанные с ее отдельными элементами. Они перечислены в табл. 11.3.

Таблица 11.3. События объекта Chart

Событие	Описание
Activate	Запускается при активизации диаграммы, например, щелчком кнопки мыши
BeforeDoubleClick	Запускается непосредственно перед действием по умолчанию при двойном щелчке на элементе диаграммы
BeforeRightClick	Запускается перед появлением контекстного меню, активизированного щелчком на элементе диаграммы правой кнопкой мыши
Calculate	Запускается при пересчете диаграммы из-за изменения исходных и добавления новых данных
Deactivate	Запускается при блокировке диаграммы при щелчке вне диаграммы кнопкой мыши
DragOver	Запускается при перетаскивании по диаграмме диапазона ячеек
DragPlot	Запускается при завершении перетаскивания по диаграмме диапазона ячеек
MouseDown	Запускается при нажатии левой или правой кнопки мыши, когда указатель располагается на диаграмме
MouseMove	Запускается при перемещении указателя мыши в пределах диаграммы
MouseUp	Запускается при отпускании левой или правой кнопки мыши, когда указатель располагается на диаграмме
Resize	Запускается при изменении размеров внедренной диаграммы или окна с диаграммой на отдельном листе
Select	Запускается, когда пользователь щелчком выбирает конкретный элемент диаграммы
SeriesChange	Запускается при перетаскивании точки данных по диаграмме

События объекта Application

В табл. 11.4 перечислены события объекта Application. Они связаны со всеми объектами в Excel.

Таблица 11.4. События объекта Application

Событие	Описание
NewWorkbook	Запускается при создании новой книги
SheetActivate	Запускается при активизации любого листа
SheetBeforeDoubleClick	Запускается при двойном щелчке на любом листе, но до стандартной реакции Excel
SheetBeforeRightClick	Запускается после щелчка на любом листе правой кнопкой мыши, но до стандартной реакции Excel
SheetCalculate	Запускается при пересчете любого листа или отражении измененных данных на диаграмме
SheetChange	Запускается при изменении ячеек любого листа
SheetPivotTableUpdate (Только в Excel 10)	Запускается после обновления листа, содержащего отчет сводной таблицы
SheetDeactivate	Запускается при деактивизации листа
SheetSelectionChange	Запускается при изменении выделенного набора на любом листе. Не возникает, если выделенный элемент находится на диаграмме
WindowActivate	Запускается при активизации окна любой книги
WindowDeactivate	Запускается при переходе к другому окну книги
WindowResize	Запускается при изменении размера окна книги
WorkbookActivate	Запускается при активизации книги
WorkbookAddinInstall	Запускается при установке книги в качестве надстройки
WorkbookAddinUninstall	Запускается при удалении рабочей книги, которая служила надстройкой
WorkbookBeforeClose	Запускается непосредственно перед закрытием любой открытой книги
WorkbookBeforePrint	Запускается перед распечаткой любой открытой книги
WorkbookBeforeSave	Запускается перед сохранением любой открытой книги
WorkbookDeactivate	Запускается при переходе к любой другой книге
WorkbookNewSheet	Запускается при создании в открытой книге нового листа
WorkbookOpen	Запускается при открытии книги
WorkbookPivotTableCloseConnection (Только в Excel 10)	Запускается после разрыва связи сводной таблицы с источником данных
WorkbookPivotTableOpenConnection (Только в Excel 10)	Запускается после установления связи сводной таблицы с источником данных

К сожалению, процесс написания кода для событий объекта Application несколько усложнен. Вот пошаговая процедура, позволяющая достичь оболочки кода:

1. С помощью меню Insert (Вставка) редактора VBA включите в проект модуль классов. Назовем этот модуль CApp (сокращенное от Class Application). В область описания модуля классов введите строку

```
Public WithEvents App As Application
```


Теперь выбор объекта App в раскрывающемся списке Object (Объект) даст доступ к оболочке кода для события объекта Application, как показано на рис. 11.2.

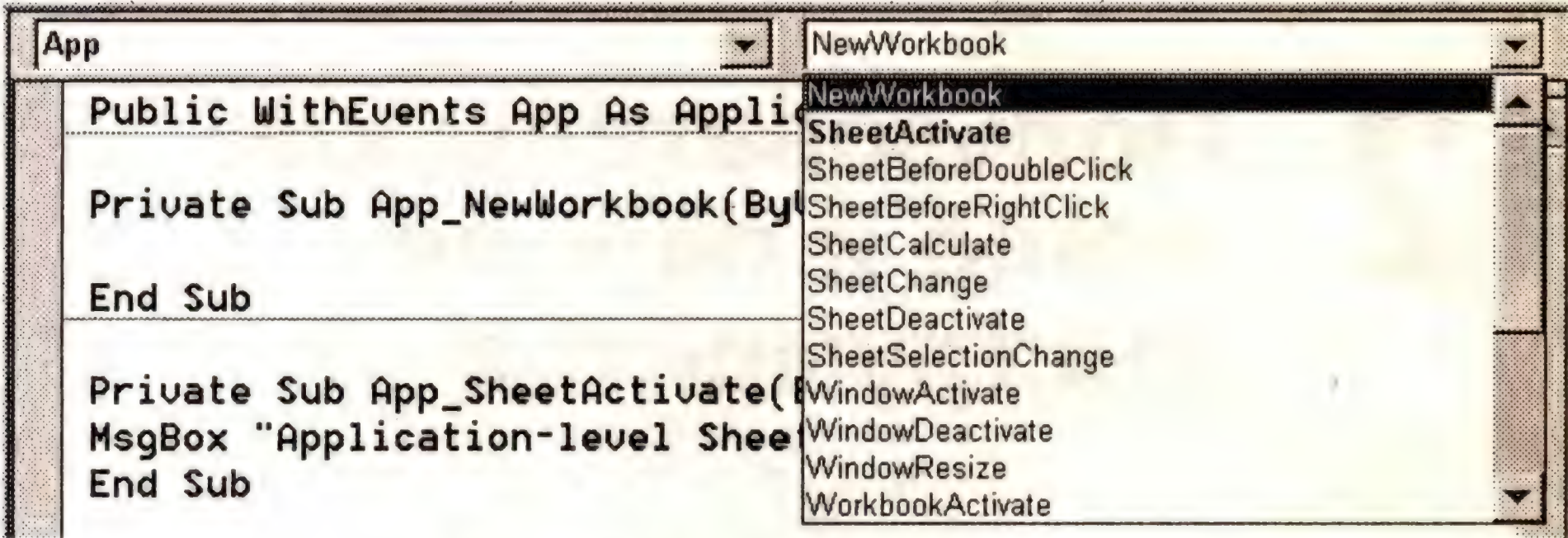


Рис. 11.2. События объекта Application

2. В модуле кода, который будет использоваться для активизации события уровня приложения, поместите следующее описание:

```
Dim AppObj As New CApp
```

3. И напоследок остается назначить свойство App объекта AppObj объекту Application. Это делается с помощью кода:

```
Set AppObj.App = Excel.Application
```

Неважно, в какое место кода вы поместите данную строку, но она должна быть выполнена, чтобы активизировать события уровня приложения. В данном случае имеет место определенное заикливание, так как этот код естественно связать с событием WorkbookOpen. Однако это событие будет выполнено только после выполнения данного кода.

Отменить использование событий уровня приложения можно как присвоив свойству EnableEvent значение False, так и указав следующую строку кода:

```
Set AppObj.App = Nothing
```


12 Пользовательские меню и панели инструментов

В этой главе будут обсуждаться методы программного управления меню и панелями инструментов. Хотя данная тема достаточно проста, у пользователей, тем не менее, периодически возникают проблемы, так как документация не настолько исчерпывающая, как хотелось бы.

Меню и панели инструментов: общий обзор

Меню и панели инструментов не относятся к объектной модели Excel. Они фигурируют во всем наборе приложений Office и соответственно принадлежат объектной модели Office. Часть этой объектной модели, связанная с меню и панелями инструментов, показана на рис. 12.1.

Наверное, вы уже заметили, что эта модель крайне невелика и содержит только два объекта и связанные с ними семейства:

- объекты `CommandBar` и семейство `CommandBars`;
- объекты `CommandBarControl` и семейство `CommandBarControls`.

Вид меню

Чтобы помочь вам освоиться с условными обозначениями, на рис. 12.2 показан компонент структуры меню Office. В данном случае это меню приложения Word, что, однако, не имеет значения.

Объект `CommandBar`

Панели инструментов, строки меню, меню, подменю и контекстные меню относятся к объектам `CommandBar`. Контекстными называются меню, появляющиеся при щелчке на объекте правой кнопкой мыши. Разумеется, панели инструментов, строки меню и контекстные меню являются объектами верхнего уровня, а дочерними для них являются меню и подменю.

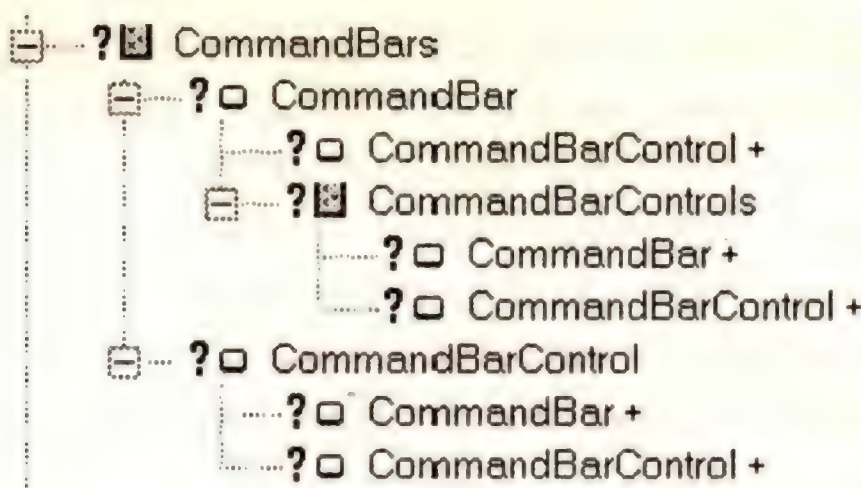


Рис. 12.1. Часть объектной модели Office, относящаяся к меню и панелям инструментов



Рис. 12.2. Меню Office

Важно понимать, что редактор VBA обрабатывает объекты CommandBar по-разному. Например, свойство Command семейства CommandBars учитывает только элементы верхнего уровня, то есть панели инструментов, строки меню и контекстные меню. Подсчет меню и подменю не производится. С помощью метода Add этого же семейства могут быть созданы панели инструментов и строки меню, но не подчиненные по отношению к ним объекты.

Объект CommandBar имеет свойство Type, которое может принимать значения констант из следующего перечисления:

```
Enum MsoBarType
    msoBarTypeNormal = 0 ' Панель инструментов
    msoBarTypeMenuBar = 1 ' Строка меню
    msoBarTypePopupMenu = 2 ' Меню, подменю или контекстное меню
End Enum
```

Элементы управления объекта CommandBar

Элементы управления панелями инструментов, строками меню, пунктами меню и подменю являются объектами CommandBarControl. Как вы вскоре убедитесь, суще-

существуют различные типы элементов управления, которые можно разделить на две категории: пользовательские (включая созданные пользователем текстовые поля, раскрывающиеся списки и т. п.) и встроенные. Имейте в виду, что эти элементы управления отличаются от используемых в объектах UserForm. Они были сконструированы специально для панелей инструментов и меню.

Существуют два специальных типа пользовательских элементов управления — это *раскрывающиеся меню* (*popup controls*) и *кнопки* (*button controls*).

Раскрывающиеся меню

Элемент управления типа `msoControlPopup` предназначен для создания раскрывающихся меню и подменю, пример которых показан на рис. 12.2. При наведении на такие элементы управления указателя мыши они принимают вид вдавленной кнопки. Их также можно идентифицировать по расположенной справа небольшой стрелке.

Итак, раскрывающийся элемент управления относится к типу `msoControlPopup` и используется для вызова меню или подменю. Раскрывающаяся командная строка относится к типу `msoBarTypePopup` и представляет собой меню, подменю или контекстное меню. Для показа раскрывающейся командной строки пользователю требуется активизировать раскрывающийся элемент управления.

Кнопки

Элементы управления типа `msoControlButton` называются кнопками. При их активизации путем щелчка левой кнопкой мыши или нажатия горячей клавиши выполняется соответствующий макрос. Кнопки обладают строковым свойством `OnAction`, которому можно присвоить имя выполняемого при активизации кнопки макроса.

Создание пункта меню

Процесс создания меню сопровождается рядом уловок. В частности, пользователи поначалу предполагают, что добавить новое меню можно с помощью метода `Add` семейства `CommandBars`, указав имя этого меню и его расположение. В конце концов, меню является объектом `CommandBar`, а описанная выше процедура применяется для добавления в семейство нового объекта.

Процесс создания нового меню представляет собой добавление элемента управления `msoControlPopup` в семейство `CommandBarControls` родительского меню и указание положения нового элемента управления по отношению к родительскому элементу. Такой подход позволяет пользователю сэкономить время. Достаточно указать, что меню или подменю активизируется с помощью раскрывающегося элемента управления, и создать этот элемент. Соответствующие пустые меню создаются автоматически. Этот процесс будет чуть позже объяснен на примере, поэтому не беспокойтесь, если вам что-то непонятно на данный момент.

При генерировании новой панели инструментов или меню можно создать данный объект временно. Это значит, что он будет уничтожен, как только окно Excel закроется. В случае непредвиденных результатов восстановить исходную ситуацию легко. Достаточно просто закрыть и снова открыть приложение Excel. Альтернативным методом удаления пункта меню является его перетаскивание за гра-

ницу меню при открытом диалоговом окне Customize (Настройка). Для удаления панели инструментов можно использовать клавишу Delete.

Семейство CommandBars

Верхним объектом иерархии для панелей инструментов и меню является семейство CommandBars, содержащее все объекты CommandBar для данного приложения. Доступ к этому семейству можно получить с помощью свойства CommandBars объекта Application, то есть написав:

```
Application.CommandBars
```

В листинге 12.1 демонстрируется код, выводящий список объектов CommandBars в окне. Вы удивитесь огромному количеству этих объектов, большинство из которых в данный момент являются невидимыми.

Листинг 12.1. Список объектов CommandBar для приложения Excel

```
Public Sub ShowCmdBars()  
  
Dim sType as string, cbar as CommandBar  
For Each cbar In Application.CommandBars  
    Select Case cbar.Type  
        Case msoBarTypeNormal      ' Панель инструментов  
            sType = "Normal"  
        Case msoBarTypeMenuBar      ' Строка меню  
            sType = "Menu bar"  
        Case msoBarTypePopup        ' Меню, подменю  
            sType = "Popup"  
    End Select  
    Debug.Print cbar.Name & ", " & sType & ", " & cbar.Visible  
Next  
  
End Sub
```

После выполнения данного кода среди множества других объектов должны оказаться:

```
Worksheet Menu Bar, Menu bar, True  
Chart Menu Bar, Menu bar, False
```

Это показывает, что главное меню выглядит по-разному для листов с данными и с диаграммами, что, впрочем, очевидно, если посмотреть на сами меню. Соответственно, если требуется добавить пользовательское меню в главную строку вне зависимости от вида активного в данный момент листа, это потребует сделать и для строки меню листа с данными, и для строки меню листа с диаграммой.

Свойство CommandBars имеет особенность, о которой нельзя не упомянуть. Если последнее связано с объектом Application, то есть указано Application.CommandBars, оно возвращает список всех встроенных и пользовательских командных панелей для рассматриваемого приложения, в данном случае для Excel. Именно поэтому в листинге 12.1 было использовано полное выражение Application.CommandBars. Помните, что для стандартного модуля кода это описание можно пропустить, указав просто CommandBars.

Однако для объекта Workbook это свойство возвращает совсем другой список. В данном случае существуют две возможности. Если книга является частью дру-

ного приложения, а Excel был активизирован двойным щелчком на ней, свойство `CommandBars` вернет набор командных панелей стороннего приложения. Элементы данного набора могут коренным образом отличаться от командных панелей Excel. Если же книга не является частью другого приложения, это свойство вернет значение `Nothing`. Более того, программного способа получить набор командных панелей, связанных с книгой, просто не существует.

То есть свойство `CommandBars` объекта `Workbook` имеет смысл только для книг, встроенных в другое приложение.

Создание новых меню и панелей инструментов

Как уже упоминалось, строки меню и панели инструментов отличаются от меню и подменю способом своего создания. Для формирования новой строки меню или нового контекстного меню достаточно воспользоваться методом `Add` из семейства `Commandbars`. Вот синтаксис этого метода:

```
CommandBarsObject.Add(Name, Position, MenuBar, Temporary)
```

Необязательный параметр `Name` задает имя новой командной панели. При пропуске данного аргумента Excel присваивает панели системное имя, например `Custom 1`. Параметр `Position` также является необязательным и указывает положение новой командной панели. Он может принимать значения `msoBarLeft`, `msoBarTop`, `msoBarRight`, `msoBarBottom`, `msoBarFloating` (для плавающей командной панели) и `msoBarPopup` (для контекстного меню).

Необязательный логический параметр `MenuBar` имеет значение `True` для строки меню и `False` для панели инструментов. По умолчанию он имеет значение `False`, то есть пропуск этого параметра приводит к появлению панели инструментов. Имейте в виду, что, создав новую строку меню и сделав ее видимой, вы заместите ею существующую строку меню. Вернуть все к исходному состоянию можно, нажав комбинацию клавиш `Alt + F4` и снова запустив приложение Excel.

Присвоение необязательному параметру `Temporary` значения `True` делает новую командную панель временной, то есть она будет удалена после закрытия окна программы. По умолчанию данный параметр имеет значение `False`.

Для иллюстрации рассмотрим код, приводящий к появлению новой плавающей панели инструментов с именем Пользовательская панель:

```
Dim cbar As Office.CommandBar
Set cbar = Application.CommandBars.Add("Пользовательская панель", _
    msoBarFloating, False, True)
cbar.Visible = True
```

Если объект `CommandBar` с именем "Пользовательская панель" уже существует, этот код приведет к появлению сообщения об ошибке "Invalid procedure call" (Неверный вызов процедуры). То есть перед использованием метода `Add` необходимо проверить существование объекта `CommandBar`, как показано в листинге 12.2.

Листинг 12.2. Создание новой панели инструментов

```
Public Sub CreateToolbar()
Dim cbar As Office.CommandBar
```



```

Dim bExists As Boolean
bExists = False
For Each cbar In Application.CommandBars
    If cbar.Name = "Custom Toolbar" Then bExists = True
Next
If Not bExists Then
    Set cbar = Application.CommandBars.Add("Пользовательская панель", _
        msoBarFloating, False, True)
    cbar.Visible = True
End If

End Sub

```

Элементы управления командной панели

Изначально одним из самых непонятных моментов системы меню Office является тот факт, что элементы строки меню представляют собой вовсе не меню и даже не их названия. На самом деле это элементы управления типа `CommandBarControl`, которые можно добавить и на другие командные панели. Они принадлежат к семейству `CommandBarControls`. Кроме того, каждый элемент управления является объектом одного из трех типов:

- `CommandBarButton`;
- `CommandBarComboBox`;
- `CommandBarPopup`.

Двойная природа объектов `CommandBarControl` дает различным элементам управления как общий, так и присущий только им набор свойств и методов. Это имеет смысл, так как, к примеру, текстовые поля существенно отличаются от раскрывающихся списков. Более того, чуть позже будет показано, что объекты `CommandBarPopup` должны обладать особым свойством (оно называется `Controls`), которое предоставляет доступ к элементам управления связанного меню. Остальным типам объектов `CommandBarControl` это свойство не требуется.

Свойство `Type` объекта `CommandBarControl` помогает определить, к какому типу данных относится элемент управления. Оно может принимать значения из следующего перечисления:

```

Enum MsoControlType
    msoControlCustom = 0
    msoControlButton = 1
    msoControlEdit = 2
    msoControlDropDown = 3
    msoControlComboBox = 4
    msoControlButtonDropDown = 5
    msoControlSplitDropDown = 6
    msoControlOCXDropDown = 7
    msoControlGenericDropDown = 8
    msoControlGraphicDropDown = 9
    msoControlPopup = 10
    msoControlGraphicPopup = 11
    msoControlButtonPopup = 12
    msoControlSplitButtonPopup = 13
    msoControlSplitButtonMRUPopup = 14

```

' <code>CommandBarButton</code>
' <code>CommandBarComboBox</code>
' <code>CommandBarComboBox</code>
' <code>CommandBarComboBox</code>
' <code>CommandBarComboBox</code>
' <code>CommandBarComboBox</code>
' <code>CommandBarComboBox</code>
' <code>CommandBarComboBox</code>
' <code>CommandBarComboBox</code>
' <code>CommandBarPopup</code>
' <code>CommandBarPopup</code>
' <code>CommandBarPopup</code>
' <code>CommandBarPopup</code>
' <code>CommandBarPopup</code>


```
msoControlLabel = 15
msoControlExpandingGrid = 16
msoControlSplitExpandingGrid = 17
msoControlGrid = 18
msoControlGauge = 19
msoControlGraphicCombo = 20      ' CommandBarComboBox
End Enum
```

Комментарии, следующие после некоторых констант, указывают тип данных соответствующего элемента управления. Эта информация находится в справочной системе Microsoft. Отсутствие комментариев означает, что этот элемент управления не принадлежит ни к одному из трех рассматриваемых типов данных.

Создание нового элемента управления командной панели

Для создания нового элемента управления командной панели используется метод Add семейства CommandBarControls. Этот метод возвращает объект CommandBarButton, CommandBarComboBox или CommandBarPopup, в зависимости от значения параметра Type. Вот его синтаксис:

```
CommandBarControlsObject.Add(Type, Id, Parameter, Before, Temporary)
```

Параметр Type задает тип элемента управления, который будет добавлен на указанную командную панель. В табл. 12.1 перечислены возможные значения этого параметра вместе с соответствующими элементами управления и типом возвращаемого методом Add значения.

Таблица 12.1. Значения параметра Type

Значение параметра	Элемент управления	Возвращаемый объект
MsoControlButton (1)	Кнопка	CommandBarButton
msoControlEdit (2)	Текстовое поле	CommandBarComboBox
msoControlDropdown (3)	Раскрывающийся список	CommandBarComboBox
msoControlComboBox (4)	Комбинированный управляющий элемент	CommandBarComboBox
msoControlPopup (5)	Раскрывающийся элемент управления	CommandBarPopup

Необязательный параметр Before является числом, которое указывает положение нового элемента управления на командной панели. При этом если в данном положении уже находится другой элемент управления, новый элемент будет вставлен перед ним. Пропуск данного параметра приводит к появлению нового элемента в конце командной панели.

Чтобы добавить пользовательский элемент управления одного из типов, перечисленных в табл. 12.1, присвойте параметру Id значение 1 или просто пропустите его. Если требуется добавить один из встроенных элементов управления, присвойте данному параметру номер этого элемента (пропустив при этом аргумент Type). Впрочем, на эту тему мы поговорим в следующем разделе.

Как и в случае с командными панелями, присвоение параметру Temporary значения True приводит к появлению временного элемента управления, который будет удален после закрытия окна программы.

Важно помнить, что объект `CommandBar` не обладает свойством `CommandBarControls`. То есть для возвращения объекта `CommandBarControls` требуется использовать свойство `Controls`, как показано ниже:

```
CommandBars("Worksheet Menu bar").Controls
```

Не менее важно осознавать, что среди всех типов объекта `CommandBarControls` свойством `Controls` обладает только один объект. Это объект `CommandBarControl` типа `CommandBarPopup`. Вышеназванное свойство дает доступ к семейству `CommandBarControls`, которое связано соответствующим меню для раскрывающихся элементов управления.

Встроенные элементы управления командной панели

Как будет показано в листинге 12.3, на панели инструментов и в меню можно поместить и встроенные элементы управления. Для этого достаточно присвоить параметру `Id` метода `Add` семейства `CommandBarControls` номер встроенного элемента управления.

Остается только один вопрос. Как определить номер встроенного элемента управления? Во-первых, можно воспользоваться методом `FindControl`. Его синтаксис выглядит следующим образом:

```
expression.FindControl(Type, Id, Tag, Visible, Recursive)
```

где `expression` представляет собой объект `CommandBar` или `CommandBars`. Все прочие параметры являются необязательными. Этот метод возвращает *первый* объект `CommandBarControl`, который удовлетворяет критерию, заданному параметрами. Если поиск оказался безуспешным, возвращается значение `Nothing`. Вот краткое описание параметров:

`Type` — одна из констант `msoControlType` перечисления, приведенного ранее в этой главе;

`Id` — номер элемента управления;

`Tag` — подпись элемента управления;

`Visible` — если данный параметр имеет значение `True`, поиск будет осуществляться только среди видимых элементов управления;

`Recursive` — присвоение данному параметру значения `True` включит в поиск командные панели и все раскрывающиеся подчиненные панели.

К сожалению, метод `FindControl` требует еще одного способа определения элемента управления, а именно указания параметра `Tag`. То есть данный метод наиболее полезен при поиске созданных вами элементов управления, которым вы лично присвоили этот параметр.

Альтернативным способом определения номера встроенного элемента управления является создание одноразового списка для дальнейших обращений. Код из листинга 12.3 создает текстовый файл и заполняет его списком названий всех встроенных элементов управления и их номерами. Имейте в виду, что для успешного выполнения кода на диске `D:` должна существовать папка `temp`. Впрочем, допустимо указать папку и маршрут доступа к ней, существующие в вашей системе. Код создает временную панель инструментов, на которую добавляет все встроенные

элементы управления с помощью цикла For и затем исследует каждый из этих элементов. Подобный подход является единственно возможным в данном случае.

Листинг 12.3. Составление списка номеров встроенных элементов управления

```
Public Sub ListControlIDs()
Dim fr As Integer
Dim cbar As Office.CommandBar
Dim ctl As CommandBarControl
Dim i As Integer
Const maxid = 4000
fr = FreeFile
Open "d:\temp\ids.txt" For Output As #fr
' Создание временной панели инструментов
Set cbar = Application.CommandBars.Add("temporary", msoBarTop, _
False, True)
For i = 1 To maxid
On Error Resume Next ' Если невозможно пропустить, добавить элемент
cbar.Controls.Add Id:=i
Next i
On Error GoTo 0
For Each ctl In cbar.Controls
Print #fr, ctl.Caption & " " & ctl.Id
Next
cbar.Delete
Close #fr
```

Листинг 12.4 демонстрирует небольшую часть файла, который получен в результате запуска вышеприведенного кода в моей системе. Полный список находится в приложении В.

Листинг 12.4. Результат выполнения кода из листинга 12.3

```
<Custom> 1
&Spelling... 2
&Save 3
&Print... 4
&New... 18
&Copy 19
Cu&t 21
&Paste 22
Open 23
Can't Repeat 37
&Microsoft Word 42
Clear Contents 47
Custom 51
&Piggy Bank 52
Custom 59
&Double Underline 60
Custom 67
Custom 68
&Close 106
AutoFormat 107
&Format Painter 108
Print Pre&view 109
Custom 112
&Bold 113
&Italic 114
&Underline 115
```


Пример использования встроенных элементов управления будет рассмотрен в конце данной главы, когда станет окончательно ясно, что они собой представляют.

Пример: создание меню

Рассмотрим код, который создает в главной строке меню систему, показанную на рис. 12.3. Обратите внимание на то, что макросы, запускаемые при выборе пунктов этого меню, называются ExampleMacro1 и ExampleMacro2.



Рис. 12.3. Пример пользовательского меню

Листинг 12.5. Создание пользовательского меню

```
Sub CreatePopup()
Dim cbpop As CommandBarControl
Dim cbctl As CommandBarControl
Dim cbsub As CommandBarControl
' Создание раскрывающегося элемента управления в строке главного меню
Set cbpop = Application.CommandBars("Worksheet Menu Bar"). _
    Controls.Add(Type:=msoControlPopup)
cbpop.Caption = "&Custom"
cbpop.Visible = True
' Добавление пункта меню
Set cbctl = cbpop.Controls.Add(Type:=msoControlButton)
cbctl.Visible = True
' Указание заголовка меню
cbctl.Style = msoButtonCaption
cbctl.Caption = "MenuItem&1"
' Действия, выполняемые при выборе пункта меню
cbctl.OnAction = "ExampleMacro1"
' Добавление раскрывающегося подменю
Set cbsub = cbpop.Controls.Add(Type:=msoControlPopup)
cbsub.Visible = True
cbsub.Caption = "&SubMenuItem1"
' Добавление пункта меню в подменю
Set cbctl = cbsub.Controls.Add(Type:=msoControlButton)
cbctl.Visible = True
' Указание заголовка меню
cbctl.Style = msoButtonCaption
cbctl.Caption = "SubMenuItem&2"
' Действия, выполняемые при выборе пункта меню
cbctl.OnAction = "ExampleMacro2"
End Sub
```

Обратите внимание на наличие знака & в свойстве Caption. Он указывает на то, что данному пункту меню соответствует клавиатурная комбинация. То есть если данное свойство записано как &MenuItem (&ПунктМеню), в строке меню появится пункт Custom (Меню пользователя), а его активизация будет производиться нажатием комбинации клавиш Alt + C.

Пример: создание панели инструментов

Сконструируем пользовательскую панель инструментов с четырьмя различными типами элементов управления, как показано на рис. 12.4. Это пример работы со встроенными элементами управления (листинг 12.6).

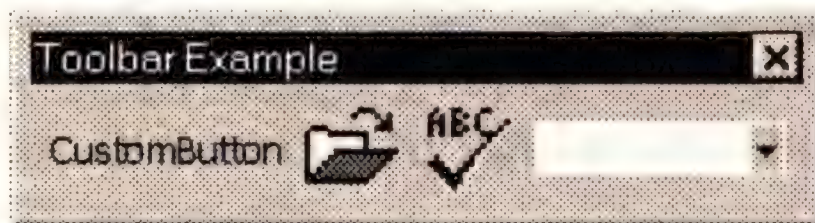


Рис. 12.4. Пользовательская панель инструментов

Листинг 12.6. Создание пользовательской панели инструментов

```
Sub CreateToolbar()
Dim cbar As CommandBar, cbctl As CommandBarControl

' Если этот элемент существует, его удаляют
For Each cbar In Application.CommandBars
    If cbar.Name = "Toolbar Example" Then cbar.Delete
Next

' Создание плавающей панели инструментов
Set cbar = Application.CommandBars.Add(Name:="Toolbar Example", _
    Position:=msoBarFloating)
cbar.Visible = True

' Добавление пользовательской кнопки для выполнения макроса
Set cbctl = cbar.Controls.Add(Type:=msoControlButton)
cbctl.Visible = True
cbctl.Style = msoButtonCaption
cbctl.Caption = "CustomButton"

' Запуск макроса
cbctl.OnAction = "ExampleMacro"

' Add built-in Open... control
Set cbctl = cbar.Controls.Add(ID:=23)

' Значок для кнопки
cbctl.FaceId = 23
cbctl.Visible = True

' Добавление кнопки для проверки орфографии
Set cbctl = cbar.Controls.Add(ID:=2)
cbctl.FaceId = 2
cbctl.Visible = True

' Добавление раскрывающегося списка
Set cbctl = cbar.Controls.Add(Type:=msoControlDropdown)

' Добавление ярлыка, который должен найти макрос
cbctl.Tag = "ComposerList"
cbctl.Visible = True
cbctl.Caption = "ListCaption"

' Задание свойств раскрывающегося списка
```



```
With cbctl
    .AddItem "Chopin", 1
    .AddItem "Mozart", 2
    .AddItem "Bach", 3
    .DropDownLines = 0
    .DropDownWidth = 75
    ' select nothing to start
    .ListIndex = 0
End With
```

```
' Макрос, который выполняется при выборе элемента
cbctl.OnAction = "ExampleListMacro"
End Sub
```

Сначала проверяется, нет ли в Excel панели инструментов с именем **Toolbar Example** (Пример панели инструментов). Если она уже существует, ее удаляют. Имя играет очень важную роль, так как позднее оно будет использовано для распознавания.

Затем добавляется пользовательская кнопка (аргумент **Id** в данном случае пропущен). Ей соответствует макрос **ExampleMacro**, код которого показан в листинге 12.7. Этот код просто сообщает о том, что кнопка была нажата.

Листинг 12.7. Макрос ExampleMacro

```
Sub ExampleMacro()
    MsgBox "Была нажата ПользовательскаяКнопка"
End Sub
```

После этого на панель был добавлен встроенный элемент управления **FileOpen**, номер которого равен 23. Нажатие этой кнопки приводит к появлению диалогового окна **Open** (Открытие документа). Обратите внимание на то, что параметру **FaceId** также присвоено значение 23. В результате для кнопки будет использован заданный по умолчанию значок, но при желании его можно поменять.

Следующим был добавлен встроенный элемент управления **Spelling**, предназначенный для проверки орфографии в активном документе.

Последним на этой панели инструментов фигурирует раскрывающийся список, который будет содержать названия трех компонентов. Учтите, что для данного элемента было указано свойство **Tag**. Это дает возможность использовать метод **FindControl** для поиска раскрывающегося списка внутри макроса, который назначен свойству **OnAction**.

Листинг 12.8. Макрос, выполняющийся при выборе элемента раскрывающегося списка

```
Sub ExampleListMacro()
    Dim cbctl As CommandBarControl
    Find the list box control
    Set cbctl = CommandBars("Toolbar Example"). _
        FindControl(Tag:="ComposerList")
    If Not cbctl Is Nothing Then
        MsgBox "Вы выбрали " & cbctl.List(cbctl.ListIndex)
    End If
End Sub
```

В настоящем макросе метод **FindControl** используется для локализации на панели инструментов элемента управления раскрывающегося списка по ярлыку этого элемента. После локализации раскрывающегося списка можно узнать, какой из

его элементов выбран. В данном примере название выбранного элемента просто появляется в информационном окне. Имейте в виду, что если критерию поиска соответствуют несколько элементов управления, показан будет первый найденный элемент. В противном случае возвращается значение `Nothing`.

Пример: новый элемент на существующем меню

Иногда вместо создания нового меню или панели инструментов пользователи предпочитают поместить элемент управления на уже существующие командные панели. В этом случае нужно включить ссылку на объект `CommandBar`, на который требуется добавить новый элемент, и использовать метод `Add` из семейства `Controls`. Этот процесс иллюстрируется в листинге 12.9, содержащем код для события `Workbook_Open`, которое добавляет пункт меню Справка по SRXUtils непосредственно перед пунктом Справка:Microsoft Excel. Обратите внимание на то, что эта процедура определяет точное расположение существующего пункта меню с помощью ссылки на объект `CommandBarController` и его свойство `Index`.

Листинг 12.9. Добавление нового элемента к существующему меню

```
Private Sub Workbook_Open()  
Dim lngPos As Long  
Dim objHelpMenu As CommandBar  
Dim objHelpMenuItem As CommandBarController  
Dim objExcelAbout As CommandBarController  
  
'Ссылка на меню Help  
Set objHelpMenu = Application.CommandBars("Help")  
  
' Определение положения элемента "Что это такое?"  
Set objExcelAbout = objHelpMenu.Controls("Что это такое?")  
If Not objExcelAbout Is Nothing Then  
    lngPos = objExcelAbout.Index  
Else  
    lngPos = objHelpMenu.Controls.Count  
End If  
  
' Добавление элемента "Справка: SRXUtils"  
Set objHelpMenuItem = objHelpMenu.Controls.Add(msoControlButton, 1, _  
    lngPos, True)  
objHelpMenuItem.Caption = "Справка: &SRXUtils"  
objHelpMenuItem.BeginGroup = True  
objHelpMenuItem.OnAction = "ShowAboutMacros"  
  
End Sub
```

Расширение приложения SRXUtils

Теперь, вооруженные сведениями о командных панелях Office, добавим новые функции надстройке SRXUtils, работу с которой вы начали в главе 10.

Создание листа с данными

По мере усложнения приложения Excel происходит усложнение связанных с ним меню. Имеет смысл хранить данные не в виде кода, а на одном из листов. Как уже упоминалось, листы надстройки скрыты от пользователя, поэтому они являются идеальным местом для хранения данных.

Откройте книгу SRXUtils.xls и удалите все листы, кроме одного. Оставшемуся листу присвойте имя DataSheet. Заполните его, как показано на рис. 12.5.

	A	B	C	D	E	F	G	H
1	Utility	OnAction Proc	Procedure	In Workbook	Menu Item	SubMenu Item	On Wks Menu	On Chart Menu
2	Activate Sheet	RunUtility	ActivateSheet	ThisWorkbook	&Activate Sheet		TRUE	TRUE
3	Print Charts	RunUtility	PrintCharts	Print.utl	&Print	Embedded &Charts	TRUE	TRUE
4	Print Pivot Tables	RunUtility	PrintPivotTables	Print.utl		&Pivot Tables	TRUE	TRUE
5	Print Sheets	RunUtility	PrintSheets	Print.utl		&Sheets	TRUE	TRUE

Рис. 12.5. Лист с данными книги SRXUtils.xls

Строки этого листа содержат данные для каждой процедуры надстройки (позднее количество строк будет увеличено). Первая строка описывает служебную программу ActivateSheet, оболочка кода для которой была создана ранее. Чуть позднее будут созданы оболочки для остальных служебных программ. В последующих главах вы займетесь их написанием.

Внимательно посмотрим на содержимое листа DataSheet. В первом столбце находятся названия служебных программ, которые используются только внутри листа. Вторым столбец содержит имя процедуры, выполняемой при активизации служебной программы с помощью пункта меню, созданного надстройкой. В данном случае все пункты меню запускают одну и ту же служебную программу RunUtility. Она определяет, какой пункт меню был выбран пользователем, и вызывает соответствующую процедуру.

В третьем столбце указывается, где именно находится процедура. Как легко убедиться, процедуры, выполняющие распечатку данных, помещены в отдельную книгу с именем Print.utl. По мере усложнения приложения может потребоваться разделить ее на несколько книг. В этом случае надстройка переписывается таким образом, чтобы файл загружался только в случае необходимости, что позволяет экономить ресурсы системы. В данном случае подобное разбиение будет сделано только для демонстрации, так как служебные программы для распечатки информации не настолько сложны, чтобы возникала необходимость помещать их в отдельную книгу.

Четвертый столбец содержит заголовки элементов меню, которые активизируют служебные программы. Обратите внимание на наличие знака &. Он указывает, что данному элементу меню соответствует некая клавиатурная комбинация. Например, элемент ActiveSheet может быть активизирован нажатием клавиши A. В пятом столбце указаны имена подменю, если таковые имеются в наличии. Например, доступ к служебной программе для распечатки данных осуществляется через подменю Print.

В последних двух столбцах указывается на наличие или отсутствие возможности работы с меню или подменю, при активном листе с данными и активном листе с диаграммой. Как уже упоминалось, в этих двух случаях вид меню будет отличаться. Например, если служебная программа предназначена для работы с диа-

граммой, совершенно незачем оставлять соответствующий ей пункт меню на панели инструментов при работе с данными.

Теперь пришло время создавать новый стандартный модуль кода, описав в нем следующие константы:

```
Public Const Utility_Col = 1
Public Const OnAction_Col = 2
Public Const Procedure_Col = 3
Public Const InWorkbook_Col = 4
Public Const MenuItem_Col = 5
Public Const SubMenuItem_Col = 6
Public Const OnWksMenu_Col = 7
Public Const OnChartMenu_Col = 8
```

Чтобы переместить любые столбцы в листе DataSheet, достаточно поменять значения данных констант.

Настройка пользовательских меню

Первым шагом в создании пользовательских меню для наших целей будет небольшое изменение кода для события Open объекта ThisWorkbook. Измените код, как показано в листинге 12.10.

Листинг 12.10. Измененная версия событий Open и Close для объекта ThisWorkbook

```
Private Sub Workbook_Open()
    CreateCustomMenus
End Sub
```

Код для создания пользовательского меню будет усложнен по сравнению с тем, который использовался в главе 10, так как требуется извлечь информацию из листа DataSheet. Это можно сделать разными способами, но в данном случае процесс будет разделен на две процедуры. Первая из них называется CreateCustomMenus и проверяет существование пользовательских меню с помощью свойства Tag. Если подобное меню уже существует, его требуется удалить. Затем вызывается вторая процедура CreateCustomMenu, которая и создает новое меню. Это делается один раз для строки меню обычного листа и один раз для строки меню листа с диаграммой. Первая процедура показана в листинге 12.11.

Листинг 12.11. Процедура CreateCustomMenus

```
Sub CreateCustomMenus()

    ' Создание пользовательского меню на листе с данными и
    ' листе с диаграммой, если оно прежде не существовало.
    ' Для идентификации меню используется свойство Tag.

    Dim cbc As CommandBarControl

    Set cbc = Application.CommandBars( _
        "Worksheet menu bar").FindControl( _
        Type:=msoControlPopup, Tag:="SRXUtilsCustomMenu")

    If Not cbc Is Nothing Then cbc.Delete
    CreateCustomMenu "Worksheet Menu Bar"

    Set cbc = Application.CommandBars( _
        "Chart menu bar").FindControl( _
```



```
Type:=msoControlPopup. Tag:="SRXUtilsCustomMenu")
If Not cbc Is Nothing Then cbc.Delete
CreateCustomMenu "Chart Menu Bar"
```

```
End Sub
```

Код процедуры CreateCustomMenu приводится в листинге 12.12. Обратите внимание на то, что свойство OnAction каждого пункта меню соответствует процедуре RunUtility, как показано в столбце OnActionProc на рис. 12.3. Она определяет, какой из пунктов меню выбран пользователем, и вызывает новую процедуру, свою для каждого конкретного случая. Чтобы передача информации в процедуру RunUtility стала возможной, были заданы свойство Tag в имени процедуры и свойство Parameter в имени содержащей эту процедуру книги. При этом для возвращения элемента управления, активизация которого приводит к выполнению процедуры RunUtility, используется свойство ActionControl данной процедуры. В результате остается только прочесть свойства Tag и Parameter элемента управления.

Листинг 12.12. Процедура CreateCustomMenu

```
Sub CreateCustomMenu(sBarName As String)

Dim cbpop As CommandBarControl
Dim cbctl As CommandBarControl
Dim cbctlCurrentPopup As CommandBarControl
Dim iEnabledColumn As Integer
Dim iLastRow As Integer
Dim iCurrentRow As Integer
Dim sCurrentMenuItem As String
Dim sCurrentSubMenuItem As String
Dim sCurrentProcedure As String
Dim sCurrentWorkbook As String
Dim sCurrentOnAction As String
Dim ws As Worksheet

iEnabledColumn = OnWksMenu_Col ' Столбец для строки меню листа с данными
If LCase(sBarName) = "chart menu bar" Then _
    iEnabledColumn = OnChartMenu_Col

Set ws = ThisWorkbook.Worksheets("DataSheet")

' Создание раскрывающегося списка sBarName на главной панели инструментов
Set cbpop = Application.CommandBars(sBarName). _
    Controls.Add(Type:=msoControlPopup, Temporary:=True)
With cbpop
    .Caption = "Cu&stom"
    .Tag = "SRXUtilsCustomMenu"
End With

' Определение последней использованной строки листа DataSheet
iLastRow = Application.WorksheetFunction.CountA(ws.Range("A:A"))

' Циклический просмотр листа DataSheet для создания пунктов меню
For iCurrentRow = 2 To iLastRow

    ' Присвоение значений
    sCurrentProcedure = ws.Cells(iCurrentRow, Procedure_Col).Value
    sCurrentWorkbook = ws.Cells(iCurrentRow, InWorkbook_Col).Value
    sCurrentMenuItem = ws.Cells(iCurrentRow, MenuItem_Col).Value
```

продолжение ➤

Листинг 12.12 (продолжение)

```

sCurrentSubMenuItem = ws.Cells(iCurrentRow, SubMenuItem_Col).Value
sCurrentOnAction = ThisWorkbook.Name & "!" & _
    ws.Cells(iCurrentRow, OnAction_Col).Value

' Если подменю отсутствует, значит, это кнопка
' в противном случае это раскрывающийся список
If sCurrentSubMenuItem = "" Then
    ' Добавление кнопки
    With cbpop.Controls.Add(Type:=msoControlButton, Temporary:=True)
        .Caption = sCurrentMenuItem
        .OnAction = sCurrentOnAction
        .Tag = sCurrentProcedure          ' Передача параметра
        .Parameter = sCurrentWorkbook    ' Передача параметра
        .Enabled = ws.Cells(iCurrentRow, iEnabledColumn).Value
    End With
Else
    ' Если раскрывающийся список еще не добавлен, сделаем это сейчас
    If sCurrentMenuItem <> "" Then
        Set cbctlCurrentPopup = cbpop.Controls.Add( _
            Type:=msoControlPopup, Temporary:=True)
        cbctlCurrentPopup.Caption = sCurrentMenuItem
    End If
    ' Теперь добавим пункт подменю
    With cbctlCurrentPopup.Controls.Add( _
        Type:=msoControlButton, Temporary:=True)
        .Caption = sCurrentSubMenuItem
        .OnAction = sCurrentOnAction
        .Tag = sCurrentProcedure          ' Передача параметра
        .Parameter = sCurrentWorkbook    ' Передача параметра
        .Enabled = ws.Cells(iCurrentRow, iEnabledColumn).Value
    End With
End If

Next      ' строка

End Sub

```

Дополнительные функции надстройки

Пришло время добавить надстройке новые функции. Как вы помните, на данный момент все функции всего лишь возвращают информационное окно со своим названием.

Благодаря служебной программе ActivateSheet в проекте SRXUtils.xls уже должен присутствовать модуль кода basMain. На данный момент этот модуль содержит следующую процедуру:

```

Public Sub ActivateSheet()
    MsgBox "Это служебная программа ActivateSheet"
End Sub

```

Для служебной программы, отвечающей за вывод на печать данных, потребуется новая книга. Создайте ее и присвойте ей имя Print.xls. Добавьте модуль кода с любым именем, воспользовавшись кодом листинга 12.13.

Листинг 12.13. Код процедуры распечатки данных

```

Public Sub PrintCharts()
    MsgBox "Это служебная программа для распечатки диаграмм"

```



```
End Sub
```

```
Public Sub PrintPivotTables()
```

```
    MsgBox "Это служебная программа для распечатки сводных таблиц"
```

```
End Sub
```

```
Public Sub PrintSheets()
```

```
    MsgBox "Это служебная программа для распечатки листов"
```

```
End Sub
```

На данный момент книга Print.xls представляет собой обычную книгу Excel и является видимой для пользователя. Впрочем, ее тоже требуется превратить в надстройку с именем Print.utl. Чтобы сохранить книгу под этим именем, введите его в кавычки в поле File name (Имя файла) диалогового окна SaveAs (Сохранить как). Если пропустить кавычки, книга будет сохранена под именем Print.utl.xla. Не забудьте выполнить обычные процедуры по созданию надстройки, то есть скомпилировать код, указать свойства книги и проекта и скрыть книгу от просмотра. Только после этого книгу можно будет сохранить в виде надстройки.

Теперь нужно поместить процедуру RunUtility, показанную в листинге 12.14, в модуль кода basMain.

Листинг 12.14. Процедура RunUtility

```
Sub RunUtility()
```

```
' С помощью свойств Tag и Parameter ищется процедура, соответствующая  
' служебной программе. Имя процедуры определяется свойством Tag.  
' а имя книги свойством Parameter.  
' Для возвращения элемента управления используется свойство ActionControl.
```

```
Dim WkbName As String
```

```
Dim ProcName As String
```

```
WkbName = Application.CommandBars.ActionControl.Parameter
```

```
If WkbName = "" Or WkbName = "ThisWorkbook" Then _
```

```
    WkbName = ThisWorkbook.Name
```

```
ProcName = Application.CommandBars.ActionControl.Tag
```

```
' Если книга закрыта, откройте ее
```

```
On Error GoTo WkbNotFound
```

```
If Not IsBookOpen(WkbName) Then
```

```
    Workbooks.Open ThisWorkbook.Path & Application.PathSeparator & WkbName
```

```
End If
```

```
' Запуск процедуры
```

```
On Error GoTo ProcNotFound
```

```
Application.Run WkbName & "!" & ProcName
```

```
Exit Sub
```

```
WkbNotFound:
```

```
    MsgBox "Невозможно найти книгу " & WkbName & " in " & _
```

```
        ThisWorkbook.Path, vbCritical, "Test Add-In"
```

```
Exit Sub
```

```
ProcNotFound:
```

```
    MsgBox "Невозможно найти процедуру " & ProcName & " in " & _
```

продолжение ➤

Листинг 12.14 (продолжение)

```
WkbName, vbCritical, "Test Add-In"
Exit Sub
```

```
End Sub
```

В листинге 12.14 осуществляется вызов функции `IsBookOpen`, код которой дан в листинге 12.15. Это делается для того, чтобы установить, открыта ли книга, содержащая процедуру. На первый взгляд это проще всего сделать перебором объектов из семейства `Workbooks`, в котором перечислены все открытые книги. Однако надстройка не только не видима пользователями, но и не фигурирует в данном семействе. К счастью, всегда остается возможность обратиться к ее книге по имени:

```
sName = Workbooks(swkbName).Name
```

Если данный код приводит к ошибке, значит, книга закрыта. В противном случае будет возвращено имя книги. Разумеется, в данный момент оно нам уже известно, но это не имеет значения.

Листинг 12.15. Функция `IsBookOpen`

```
Private Function IsBookOpen(swkbName) As Boolean
```

```
' Проверяем, открыта ли книга
' Так как книга надстройки не видна в семействе
' Workbooks, требуется другой метод.
' Однако можно сослаться на надстройку по ее имени, поэтому
' достаточно получить доступ к свойству Name.
' Если появится сообщение об ошибке, книга закрыта.
```

```
Dim sName As String
On Error GoTo WkbNotOpen
IsBookOpen = True
sName = Workbooks(swkbName).Name
```

```
Exit Function
```

```
WkbNotOpen:
IsBookOpen = False
```

```
End Function
```

Завершение работы с надстройкой

Чтобы закрыть книгу `SRXUtils.xla`, достаточно снять флажок напротив ее имени в диалоговом окне `Add-Ins (Надстройки)`. Однако это не поможет закрыть надстройки, расположенные в коде, такие как `Print.utl`. Закрыть все открытые надстройки можно с помощью события `BeforeClose`, которое на данный момент отвечает только за удаление пользовательских меню.

Существует простой, хотя и не элегантный подход. Можно закрыть все надстройки, перечисленные на листе `DataSheet`, кроме книги `SRXUtils.xla`. Эта книга закрывается, когда пользователь снимает флажок напротив ее имени в диалоговом окне `Add-Ins (Надстройки)`. В этом случае требуется строка `On Error Resume Next`, благодаря которой игнорируются попытки закрыть неоткрытую книгу. Соответственно, код события `BeforeClose` должен быть изменен, как показано в листинге 12.16.

Листинг 12.16. Код обработки события Workbook_BeforeClose

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)

' Удаление пользовательских меню и закрытие всех надстроек

Dim r As Integer
Dim ws As Worksheet
Dim sName As String

' В случае попытки закрыть книгу, которая не была открыта
On Error Resume Next

DeleteCustomMenus

Set ws = ThisWorkbook.Worksheets("DataSheet")
For r = 2 To Application.WorksheetFunction.CountA(ws.Range("A:A"))
    sName = ws.Cells(r, InWorkbook_Col).Value
    If sName <> "" And sName <> "ThisWorkbook" Then
        Workbooks(sName).Close
    End If
Next r

End Sub
```

Код процедуры DeleteCustomMenu показан в листинге 12.17.

Листинг 12.17. Процедура DeleteCustomMenu

```
Sub DeleteCustomMenus()

Dim cbc As CommandBarControl

Set cbc = Application.CommandBars("Worksheet menu bar")._
FindControl(Type:=msoControlPopup, Tag:="TestAddInCustomMenu")
If Not cbc Is Nothing Then cbc.Delete

Set cbc = Application.CommandBars("Chart menu bar")._
FindControl(Type:=msoControlPopup, Tag:="TestAddInCustomMenu")
If Not cbc Is Nothing Then cbc.Delete

End Sub
```

Теперь все необходимые фрагменты программы написаны, и вы можете сохранить файл SRXUtils.xls в виде надстройки, соблюдая порядок действий, описанный в главе 10. В случае возникновения проблем можно загрузить код данной надстройки с веб-сайта издательства O'Reilly и сравнить с вашей версией кода.

13 Встроенные диалоговые окна

В объектной модели Excel существует объект Dialog для каждого из встроенных диалоговых окон. Эти объекты формируют семейство Dialogs и соответствуют константам перечисления XlBuiltInDialog, как показано в табл. 13.1 и 13.2. Семейство Dialogs возвращается свойством Dialogs объекта Application. В табл. 13.1 знак <vX> указывает на то, что константа впервые появилась в версии X.

Таблица 13.1. Константы и значения перечисления XlBuiltInDialog

_xlDialogChartSourceData (541)<v9>	xlDialogFormatMove (128)	xlDialogPrinterSetup (9)
_xlDialogPhonetic (538)<v9>	xlDialogFormatNumber (42)	xlDialogPrintPreview (222)
xlDialogActivate (103)	xlDialogFormatOverlay (226)	xlDialogPromote (202)
xlDialogActiveCellFont (476)	xlDialogFormatSize (129)	xlDialogProperties (474)
xlDialogAddChartAutoformat (390)	xlDialogFormatText (89)	xlDialogPropertyFields (754)<v10>
xlDialogAddinManager (321)	xlDialogFormulaFind (64)	xlDialogProtectDocument (28)
xlDialogAlignment (43)	xlDialogFormulaGoto (63)	xlDialogProtectSharing (620)
xlDialogApplyNames (133) (653)<v9>	xlDialogFormulaReplace (130)	xlDialogPublishAsWebPage
xlDialogApplyStyle (212)	xlDialogFunctionWizard (450)	xlDialogPushbuttonProperties (445)
xlDialogAppMove (170)	xlDialogGallery3dArea (193)	xlDialogReplaceFont (134)
xlDialogAppSize (171)	xlDialogGallery3dBar (272)	xlDialogRoutingSlip (336)
xlDialogArrangeAll (12)	xlDialogGallery3dColumn (194)	xlDialogRowHeight (127)
xlDialogAssignToObject (213)	xlDialogGallery3dLine (195)	xlDialogRun (17)
xlDialogAssignToTool (293)	xlDialogGallery3dPie (196)	xlDialogSaveAs (5)
xlDialogAttachText (80)	xlDialogGallery3dSurface (273)	xlDialogSaveCopyAs (456)
xlDialogAttachToolbars (323)	xlDialogGalleryArea (67)	xlDialogSaveNewObject (208)
xlDialogAutoCorrect (485)	xlDialogGalleryBar (68)	xlDialogSaveWorkbook (145)
xlDialogAxes (78)	xlDialogGalleryColumn (69)	xlDialogSaveWorkspace (285)
xlDialogBorder (45)	xlDialogGalleryCustom (388)	xlDialogScale (87)
xlDialogCalculation (32)	xlDialogGalleryDoughnut (344)	xlDialogScenarioAdd (307)
xlDialogCellProtection (46)	xlDialogGalleryLine (70)	xlDialogScenarioCells (305)
xlDialogChangeLink (166)	xlDialogGalleryPie (71)	xlDialogScenarioEdit (308)
xlDialogChartAddData (392)	xlDialogGalleryRadar (249)	xlDialogScenarioMerge (473)

xlDialogChartLocation (527)		xlDialogGalleryScatter (72)
xlDialogScenarioSummary (311)		
xlDialogChartOptionsData- LabelMultiple (724)<v10>	xlDialogGoalSeek (198)	xlDialogScrollbarProperties (420)
xlDialogChartOptionsDataLabels (505)	xlDialogGridlines (76)	xlDialogSearch (731)<v10>
xlDialogChartOptionsDataTable (506)	xlDialogImportTextFile (666)<v9>	xlDialogSelectSpecial (132)
xlDialogChartSourceData (540)	xlDialogInsert (55)	xlDialogSendMail (189)
xlDialogChartTrend (350)	XlDialogInsertHyperlink (596)	xlDialogSeriesAxes (460)
xlDialogChartType (526)	xlDialogInsertNameLabel (496)	xlDialogSeriesOptions (557)
xlDialogChartWizard (288)	xlDialogInsertObject (259)	xlDialogSeriesOrder (466)
xlDialogCheckboxProperties (435)	xlDialogInsertPicture (342)	xlDialogSeriesShape (504)
xlDialogClear (52)	xlDialogInsertTitle (380)	xlDialogSeriesX (461)
xlDialogColorPalette (161)	XlDialogLabelProperties (436)	xlDialogSeriesY (462)
xlDialogColumnWidth (47)	xlDialogListboxProperties (437)	xlDialogSetBackgroundPicture (509)
xlDialogCombination (73)	xlDialogMacroOptions (382)	xlDialogSetPrintTitles (23)
xlDialogConditionalFormatting (583)	xlDialogMailEditMailer (470)	xlDialogSetUpdateStatus (159)
xlDialogConsolidate (191)	xlDialogMailLogon (339)	xlDialogShowDetail (204)
xlDialogCopyChart (147)	xlDialogMailNextLetter (378)	xlDialogShowToolbar (220)
xlDialogCopyPicture (108)	xlDialogMainChart (85)	xlDialogSize (261)
xlDialogCreateNames (62)	XlDialogMainChartType (185)	xlDialogSort (39)
xlDialogCreatePublisher (217)	xlDialogMenuEditor (322)	xlDialogSortSpecial (192)
xlDialogCustomizeToolbar (276)	xlDialogMove (262)	xlDialogSplit (137)
xlDialogCustomViews (493)	xlDialogNew (119)	xlDialogStandardFont (190)
xlDialogDataDelete (36)	xlDialogNewWebQuery (667)<v9>	xlDialogStandardWidth (472)
xlDialogDataLabel (379)	xlDialogNote (154)	xlDialogStyle (44)
xlDialogDataLabelMultiple (723)<v10>	xlDialogObjectProperties (207)	xlDialogSubscribeTo (218)
xlDialogDataSeries (40)	xlDialogObjectProtection (214)	xlDialogSubtotalCreate (398)
xlDialogDataValidation (525)	xlDialogOpen (1)	xlDialogSummaryInfo (474)
xlDialogDefineName (61)	xlDialogOpenLinks (2)	xlDialogTable (41)
xlDialogDefineStyle (229)	xlDialogOpenMail (188)	xlDialogTabOrder (394)
xlDialogDeleteFormat (111)	xlDialogOpenText (441)	xlDialogTextToColumns (422)
xlDialogDeleteName (110)	xlDialogOptionsCalculation (318)	xlDialogUnhide (94)
xlDialogDemote (203)	xlDialogOptionsChart (325)	xlDialogUpdateLink (201)
xlDialogDisplay (27)	xlDialogOptionsEdit (319)	xlDialogVbaInsertFile (328)
xlDialogEditboxProperties (438)	XlDialogOptionsGeneral (356)	xlDialogVbaMakeAddin (478)
xlDialogEditColor (223)	xlDialogOptionsListsAdd (458)	XlDialogVbaProcedureDefinition (330)
xlDialogEditDelete (54)	xlDialogOptionsME (647)<v9>	xlDialogView3d (197)
xlDialogEditionOptions (251)	xlDialogOptionsTransition (355)	xlDialogWebOptionsBrowsers (773)<v10>
xlDialogEditSeries (228)	xlDialogOptionsView (320)	xlDialogWebOptionsEncoding (686)<v9>
xlDialogErrorbarX (463)	xlDialogOutline (142)	xlDialogWebOptionsFiles (684)<v9>
xlDialogErrorbarY (464)	xlDialogOverlay (86)	xlDialogWebOptionsFonts (687)<v9>

продолжение ↗

Таблица 13.1. (продолжение)

xlDialogErrorChecking (683)<v9>	xlDialogOverlayChartType (186)	xlDialogWebOptionsGeneral (732)<v10>
xlDialogEvaluateFormula (709)<v10>	xlDialogPageSetup (7)	xlDialogWebOptionsPictures (685)<v9>
xlDialogExternalDataProperties (530)<v9>	xlDialogParse (91)	xlDialogWindowMove (14)
xlDialogExtract (35)	xlDialogPasteNames (58)	xlDialogWindowSize (13)
xlDialogFileDelete (6)	xlDialogPasteSpecial (53)	xlDialogWorkbookAdd (281)
xlDialogFileSharing (481)	xlDialogPatterns (84)	xlDialogWorkbookCopy (283)
xlDialogFillGroup (200)	xlDialogPhonetic (656)	xlDialogWorkbookInsert (354)
xlDialogFillWorkgroup (301)	xlDialogPivotCalculatedField (570)	xlDialogWorkbookMove (282)
xlDialogFilter (447)	xlDialogPivotCalculatedItem (572)	xlDialogWorkbookName (386)
xlDialogFilterAdvanced (370)	xlDialogPivotClientServerSet (689)<v9>	xlDialogWorkbookNew (302)
xlDialogFindFile (475)	xlDialogPivotFieldGroup (433)	xlDialogWorkbookOptions (284)
xlDialogFont (26)	xlDialogPivotFieldProperties (313)	xlDialogWorkbookProtect (417)
xlDialogFontProperties (381)	xlDialogPivotFieldUngroup (434)	xlDialogWorkbookTabSplit (415)
xlDialogFormatAuto (269)	xlDialogPivotShowPages (421)	xlDialogWorkbookUnhide (384)
xlDialogFormatChart (465)	xlDialogPivotSolveOrder (568)	xlDialogWorkgroup (199)
xlDialogFormatCharttype (423)	xlDialogPivotTableOptions (567)	xlDialogWorkspace (95)
xlDialogFormatFont (150)	xlDialogPivotTableWizard (312)	xlDialogZoom (256)
xlDialogFormatLegend (88)	xlDialogPlacement (300)	
xlDialogFormatMain (225)	xlDialogPrint (8)	

Таблица 13.2. Дополнительные константы XlBiultInDialog и их значения в Excel 9.0

_xlDialogChartSourceData (541)	xlDialogOptionsME (647)	xlDialogWebOptionsFonts (687)
_xlDialogPhonetic (538)	xlDialogPivotClientServerSet (689)	xlDialogWebOptionsGeneral (683)
xlDialogExternalDataProperties (530)	xlDialogPublishAsWebPage (653)	xlDialogWebOptionsPictures (685)
xlDialogImportTextFile (666)	xlDialogWebOptionsEncoding (686)	
xlDialogNewWebQuery (667)	xlDialogWebOptionsFiles (684)	

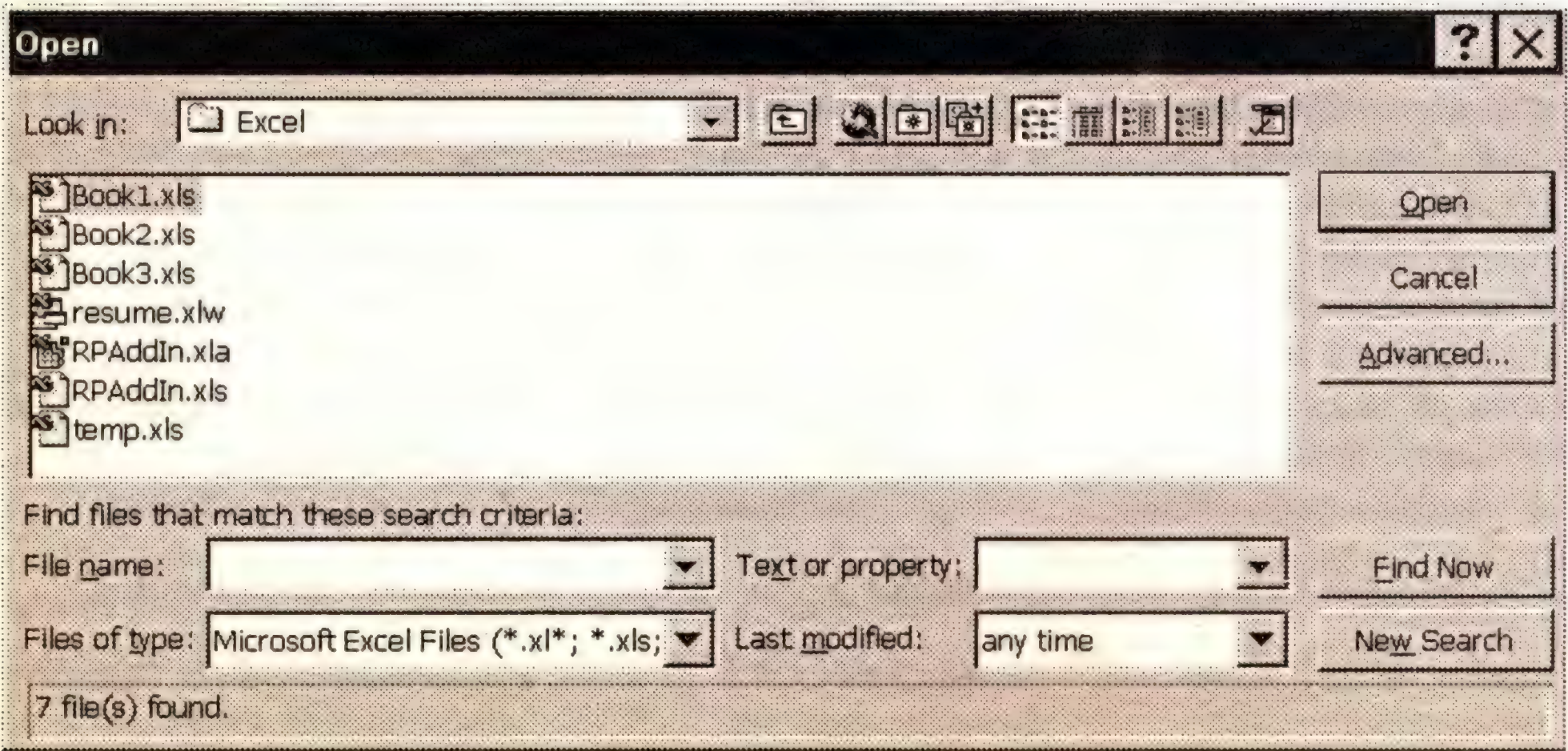


Рис. 13.1. Диалоговое окно Open

Обратите внимание: в названии каждой из констант в табл. 13.1 фигурирует приставка `xlDialog`, за которой следует имя диалогового окна. Например, константа, соответствующая диалоговому окну `Open` (Открытие документа), называется `xlDialogOpen`, а соответствующий объект `Dialog` будет выглядеть так:

```
Application.Dialogs(xlDialogOpen)
```

Вид диалогового окна `Open` (Открытие документа) показан на рис. 13.1.

К сожалению, объект `Dialog` имеет только один полезный метод — `Show`.

Метод Show

Метод `Show` приводит к появлению диалогового окна. Это обеспечивает пользователю удобный способ работы со встроенными диалоговыми окнами. К сожалению, доступ к значениям, которые пользователь вводит в диалоговое окно, получить невозможно. Контроль над цепью событий возвращается только после того, как диалоговое окно будет закрыто, а указанный с его помощью набор действий выполнен. Хотя в некоторых приложениях, например в Word 97, не предусмотрена автоматическая обработка значений, которые пользователь вводит во встроенные диалоговые окна.

Для иллюстрации рассмотрим код:

```
Application.Dialogs(xlDialogOpen).Show
```

вызывающий диалоговое окно, изображенное на рис. 13.1. Метод `Show` возвращает значение `True`, если пользователь щелкает на кнопке `OK`, и значение `False` в случае щелчка на кнопке `Cancel`. В первом случае происходит выполнение действий с параметрами, которые были введены в текстовые поля. В случае с окном `Open` (Открытие документа) это значит, что выбранный файл будет открыт приложением Excel. В то же время если будет нажата кнопка `Cancel`, никаких действий предпринято не будет.

Метод `Show` имеет следующий синтаксис:

```
DialogObject.Show(arg1, arg2, .... arg30)
```

Аргументы в данном случае используются для задания параметров диалогового окна. Перечень этих аргументов можно найти в справочном файле VBA. Например, диалоговое окно `xlDialogOpen` имеет следующий перечень аргументов:

```
file_text  
update_links  
read_only  
format  
prot_pwd  
write_res_pwd  
ignore_rorec  
file_origin  
custom_delimit  
add_logical  
editable  
file_access  
notify_logical  
converter
```


Следовательно, код

```
Application.Dialogs(xlDialogOpen).Show "*.*", False, True
```

приведет к появлению несколько модифицированного диалогового окна Open (Открытие документа). Раскрывающийся список File of Type (Тип файлов) будет по умолчанию показывать значение All files «*.»* (Все файлы «*.»*»). Параметр update_links примет значение False. В результате автоматического обновления ссылок в Excel не происходит. Параметр read_only будет иметь значение True, соответственно, пользователь потеряет возможность вносить изменения в открываемые файлы.

К сожалению, в документации, предложенной фирмой Microsoft, отсутствуют пояснения по поводу значений различных аргументов. Кроме того, эти аргументы не являются именованными, то есть если вы не хотите указывать значение последних, их требуется заменять пробелами. В результате единственным способом настройки параметров встроенных диалоговых окон является метод проб и ошибок.

14

Нестандартные диалоговые окна

Как вы уже успели убедиться, встроенные диалоговые окна дают крайне ограниченную возможность передачи информации от пользователя к приложению. К счастью, Excel позволяет создавать нестандартные диалоговые окна с более гибкими средствами передачи. Они называются *формами (form)*, или объектами UserForm. В этой главе вы познакомитесь с основными приемами создания данных объектов, в результате чего будет заложен фундамент для более подробного знакомства с настоящей темой.

Впрочем, для большинства приложений Excel требуются достаточно простые формы. Например, часто используются формы с текстовыми полями для ввода текста, с раскрывающимися списками для выбора одного варианта из предложенного набора или с переключателем для выбора одного параметра. Разумеется, форма должна содержать и командные кнопки, позволяющие выполнять процедуры.

Редактор Visual Basic хорошо подходит для создания приложений со сложными формами. Впрочем, он был разработан именно для этой цели. Вдобавок указанный редактор, как и приложение Excel, дает доступ к набору объектных моделей Microsoft Office.

Объект UserForm

Объект UserForm можно определить как стандартный модуль кода с визуальным интерфейсом, который используется для взаимодействия с пользователем. Впрочем, это описание не стоит понимать слишком буквально. Например, процедуры (даже уровня проекта), описанные в общей области модуля UserForm, обычно поддерживают объекты (или код) только самой формы, в то время как процедуры уровня проекта, определенные в стандартном модуле, поддерживают код, расположенный в любом месте проекта, а не только в их собственном модуле.

Для иллюстрации вышесказанного предположим, что в общей области модуля UserForm, называемой UserForm1, описана процедура ProcedureA. Несмотря на то что это процедура уровня проекта, получить к ней доступ из другого модуля, просто написав:

```
ProcedureA.
```


невозможно. В то же время если бы процедура была определена в стандартном модуле, это оказалось бы вполне допустимым. Но в данном случае придется воспользоваться полным именем:

```
UserForm1.ProcedureA
```

Создание объекта UserForm

Для создания нестандартного диалогового окна в режиме конструирования достаточно выделить название проекта, в котором оно должно фигурировать, а затем выбрать команду UserForm в меню Insert редактора VBA. В режиме выполнения создание формы осуществляется с помощью метода Add, но в данной книге этот процесс будет ограничен режимом конструирования. На рис. 14.1 показано, как выглядит среда конструирования после выбора объекта UserForm в окне Project.

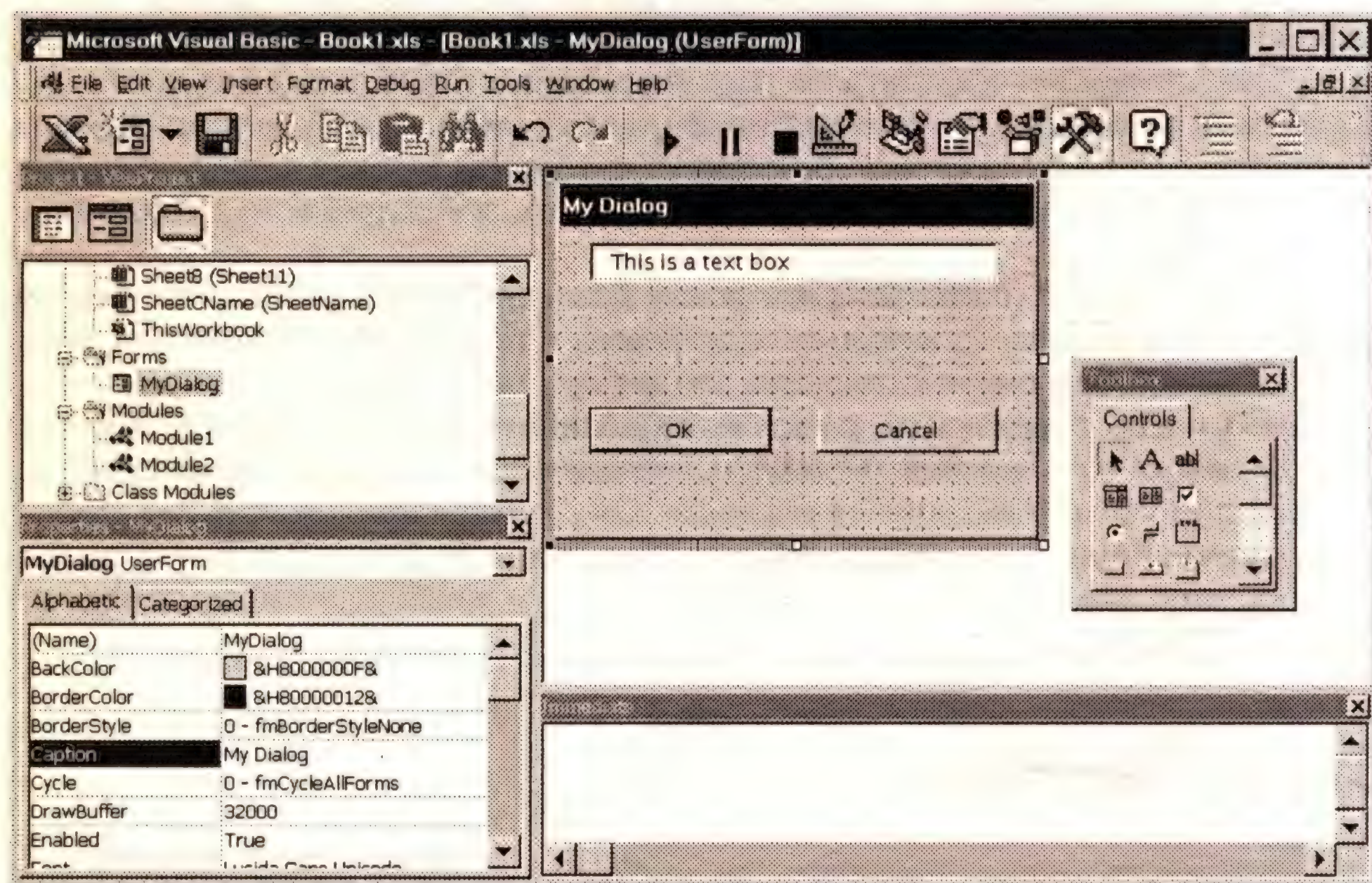


Рис. 14.1. Вид диалогового окна UserForm в режиме конструирования

В правом окне на рис. 14.1 показано диалоговое окно с текстовым полем и двумя командными кнопками. Рядом располагается панель инструментов Toolbox со значками различных элементов управления.

Чтобы поместить в форму элемент управления, достаточно щелкнуть на соответствующем значке панели инструментов Toolbox и затем нарисовать прямоугольник на форме. Когда вы отпустите кнопку мыши, на месте прямоугольника появится нужный элемент управления. Для изменения свойств формы или любого ее элемента нужно выделить объект и ввести новую информацию в поля окна Properties.

Выбор в меню Tools редактора VBA команды Additional Controls приводит к появлению диалогового окна с перечнем дополнительных элементов управления.

Элемент управления ActiveX

Те, кто достаточно долго проработал в операционной системе Microsoft Windows (предполагается, что это относится и к вам, так как вы читаете данную книгу), хорошо знакомы с элементами управления пользовательского уровня. Сюда относятся:

- командные кнопки;
- текстовые поля;
- раскрывающиеся списки;
- поля со списком;
- переключатели;
- флажки;
- надписи;
- вкладки;
- полосы прокрутки.

Для удобства пользователя перечисленные элементы управления снабжены визуальным интерфейсом. Однако существуют и элементы, у которых он отсутствует. Например, к таковым относится элемент управления Timer, который вызывает события через определенные промежутки времени.

В сущности, элемент управления представляет собой специальный тип компонента кода, помещаемый в некий объект-контейнер (например, в форму). Он имеет следующие свойства:

- в большинстве случаев у него имеется визуальный интерфейс для удобства работы пользователей;
- у него существует набор методов, активизируемых пользователями;
- у него имеется набор свойств, которые читаются и редактируются пользователями;
- он связан с некими событиями, код для которых пишется пользователями.

События, связанные с объектами Excel, рассматривались в главе 11. Как вы вскоре убедитесь, события, связанные с элементами управления, работают по той же схеме.

Код объекта UserForm

В общем случае программист добавляет в модуль UserForm два варианта кода: код события, связанного с различными элементами управления формы (а в некоторых случаях и с самой формой), и код дополнительных процедур, выполняющих функции, требуемые в приложении. Последний вариант кода добавляется в раздел общих описаний модуля кода UserForm.

Проиллюстрируем вышесказанное на простом примере. Предположим, что требуется создать приложение, сортирующее выделенные столбцы (при этом каждый столбец рассматривается как единый объект) по содержимому первого столбца. Возможный вид формы для данной операции показан на рис. 14.2.

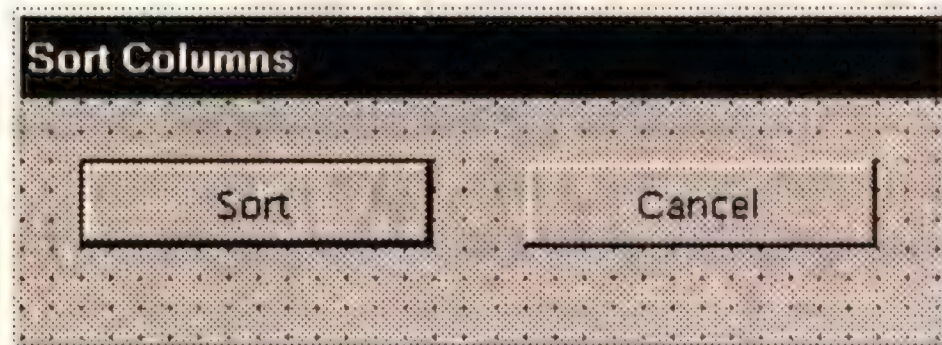


Рис. 14.2. Диалоговое окно для сортировки

В результате щелчка на кнопке Сортировать появляется информационное окно с вопросом, действительно ли нужно осуществить сортировку, затем срабатывает Click для данной кнопки. Например, для кнопки с именем cmdSort редактор поддерживает следующую оболочку кода:

```
Private Sub cmdSort_Click()
```

```
End Sub
```

Понятно, что при наступлении этого события должна выполняться операция сортировки. Однако помещать код сортировки в оболочку кода события не очень хорошая идея. Вместо этого требуется написать отдельную процедуру сортировки, которая будет находиться в общей области модуля UserForm. Или же ее можно сделать процедурой уровня проекта, расположив в отдельном стандартном модуле кода:

```
Public Sub SortColumns()
```

```
' Здесь находится код для сортировки текста
```

```
End Sub
```

Вот причины, по которым код сортировки лучше поместить в отдельный модуль:

- этот код можно будет использовать в других местах приложения;
- упростится процесс перемещения кода в другое приложение;
- поиск и исправление ошибок также станет более простым делом;
- упростится редактирование кода;
- улучшится читаемость кода.

После завершения процедуры сортировки с событием Click можно связать следующий код:

```
Private Sub cmdSort_Click()
```

```
If MsgBox("Сортировать выделенные столбцы?",  
vbQuestion + vbYesNo) = vbYes Then SortColumns
```

```
End Sub
```

Событию Click для кнопки Cancel в большинстве случаев соответствует следующий код:

```
Private Sub cmdCancel_Click()
```

```
Unload Me
```

```
End Sub
```


Этот код приводит к тому, что изображение формы пропадает с экрана.

Важно понимать разницу между выгруженной и скрытой формами. Скрыть форму можно, присвоив свойству `Visible` значение `False`. При этом на поддержание невидимой формы все равно тратятся ресурсы компьютера. Если же форма выгружена, а всем указывающим на нее ссылкам присвоено значение `Nothing`, ресурсы на ее поддержание не тратятся.

Стандартные элементы управления Excel

В Excel существуют два типа элементов управления. На рис. 14.3 показаны две панели инструментов, каждая из которых дает доступ к одному типу элементов управления. Под каждой из них показано, как выглядят создаваемые с их помощью кнопки.

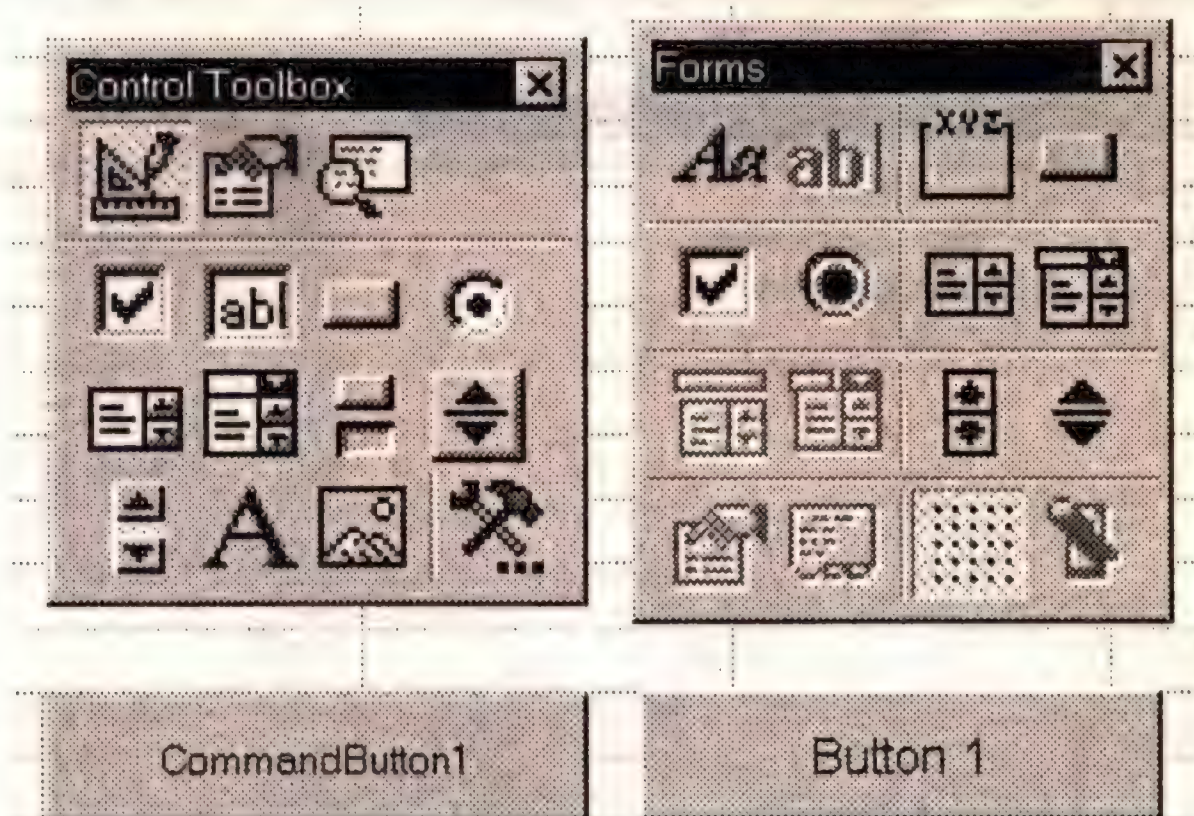


Рис. 14.3. Панели инструментов различных элементов управления

Панель инструментов Control Toolbox (Элементы управления) (она показана слева) отвечает за создание элементов управления ActiveX, которые помещаются на объект UserForm или непосредственно на лист (но не на диаграмму). Тот же самый набор инструментов находится на панели Toolbox, применяемой в редакторе VBA для конструирования объектов UserForm. Элементы управления ActiveX достаточно гибки и поддерживают широкий спектр событий. Для вызова панели инструментов Control Toolbox (Элементы управления) достаточно установить флажок рядом с названием этой панели на вкладке Toolbars (Панели инструментов) диалогового окна Customize (Настройка). Это окно вызывается одноименной командой меню Tools (Сервис).

Обратите внимание на то, как различается вид панелей инструментов, используемых для создания элементов управления ActiveX, на рис. 14.1 и 14.3. Панель инструментов Toolbox применяется при работе с формами, а панель инструментов Control Toolbox (Элементы управления) — при работе с листами. На панели Control Toolbox (Элементы управления) самой важной является кнопка Design Mode (Режим конструктора). Щелчок на ней приводит к переходу в *режим конструирования*.

ния (*design mode*), в котором перемещать и менять размер элементов управления на листе можно с помощью мыши. Щелчок правой кнопкой мыши на элементе управления приводит к появлению диалогового окна с параметрами этого элемента. Если же кнопка Design Mode (Режим конструктора) не нажата, щелчок на любом элементе управления просто запускает событие Click.

Чтобы открыть панель инструментов Forms (Формы), показанную справа на рис. 14.3, нужно установить флажок рядом с ее именем в списке Toolbars (Панели инструментов) диалогового окна Customize (Настройка). Эти элементы управления называются стандартными и остаются неизменными с версии Excel 5.0. Они размещаются на листах с данными или с диаграммой и запускают исключительно событие Click.

Например, при создании на листе стандартной кнопки немедленно отобразится диалоговое окно Assign Macro (Назначить макрос объекту), показанное на рис. 14.4. Оно позволяет связать с событием Click некий макрос.

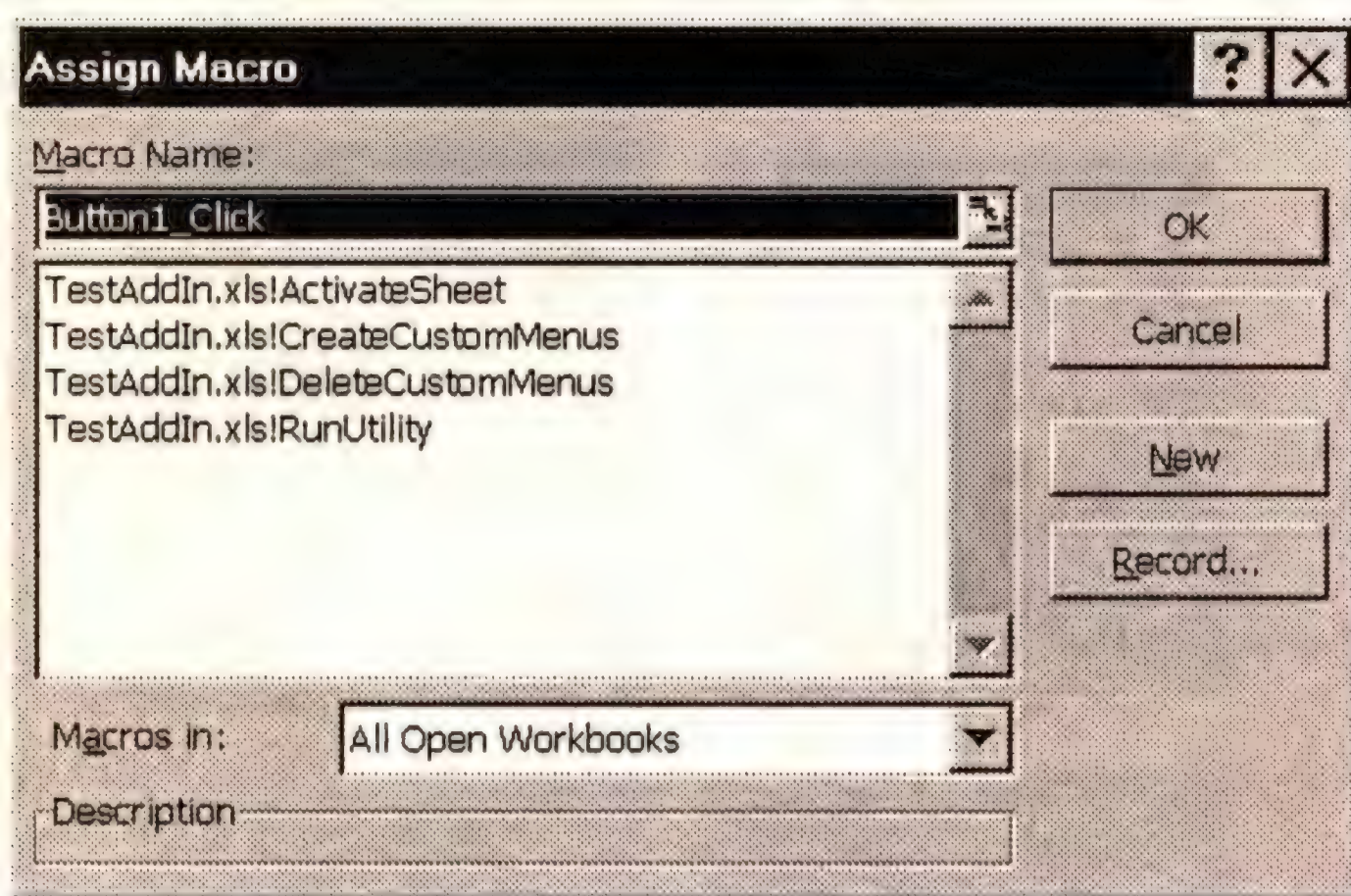


Рис. 14.4. Ответ на размещение на листе стандартной кнопки Excel

Стандартные элементы управления до сих пор повсеместно используются, так как только их можно размещать на диаграммах Excel. В остальных случаях следует применять более гибкие элементы управления ActiveX.

Пример: служебная программа ActivateSheet

Пришло время обогатить функциональные свойства служебной программы ActivateSheet приложения SRXUtils и проиллюстрировать работу с объектами UserForm.

В частности, при активизации данной процедуры должно появляться диалоговое окно со списком всех листов открытой книги, как показано на рис. 14.5. Затем пользователь выбирает лист, который он хотел бы активизировать.

Процесс доработки этой служебной программы выглядит следующим образом:

- изменить процедуру ActivateSheet в модуле basMain таким образом, чтобы она открывала диалоговое окно Активизация листа вместо информационного окна с сообщением;

- сконструировать окно Активизация листа;
- написать код, соответствующий этому диалоговому окну.

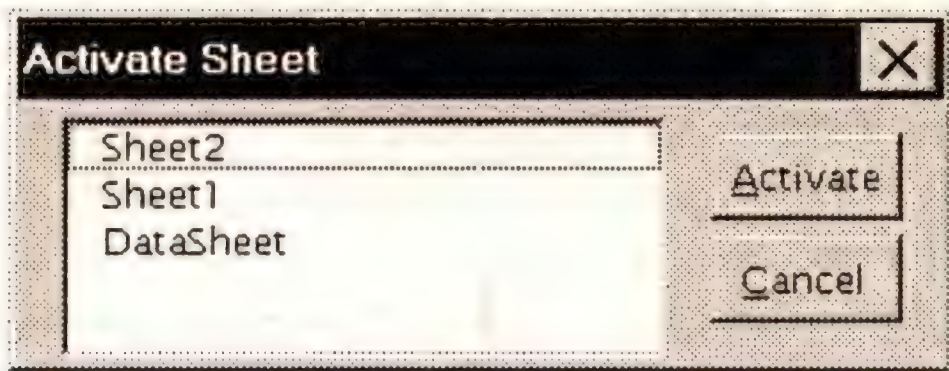


Рис. 14.5. Диалоговое окно, предназначенное для активизации листов

Изменение процедуры ActivateSheet

Итак, вернемся к листу SRXUtils.xls и изменим процедуру ActivateSheet в модуле basMain:

```
Public Sub ActivateSheet()  
    MsgBox "Это служебная программа ActivateSheet"  
End Sub
```

на процедуру:

```
Public Sub ActivateSheet()  
    dlgActivateSheet.Show  
End Sub.
```

которая приводит к появлению диалогового окна Activate Sheet (Активизация листа). Мы будем называть его dlgActivateSheet.

Создание объекта UserForm

После включения в проект объекта UserForm требуется присвоить свойству Name в окне Properties значение dlgActivateSheet, а свойству Caption — значение Активизация листа. Затем в объект можно добавить элементы управления. Диалоговое окно на рис. 14.5 имеет две кнопки и одно поле со списком.

Поле со списком

Поместите в форму UserForm поле со списком, как показано на рис. 14.5. Для этого нужно щелкнуть на кнопке ListBox панели инструментов Toolbox и нарисовать в форме прямоугольную область. В окне Properties настройте свойства, перечисленные в табл. 14.1. Имейте в виду, что свойство TabIndex определяет не только порядок, в котором курсор переходит на элементы управления при нажатии клавиши Tab, но и элемент управления, на котором курсор находится изначально. Так как изначально требуется сфокусировать внимание на поле со списком, этому свойству присваивается значение 0.

Таблица 14.1. Свойства элемента управления ListBox

Свойство	Значение
Name	LstSheets
TabIndex	0

Напомним, что в общем случае для задания свойств можно использовать как окно `Properties` во время конструирования, так и код во время исполнения. Некоторые свойства задаются исключительно в режиме конструирования, в то время как другие — исключительно в режиме исполнения. Впрочем, большинство свойств не имеет подобных ограничений.

Например, свойства `Visible` или `Enable` часто настраиваются при запуске процедуры как ответ на действия пользователя. Скажем, можно заблокировать командную кнопку с ярлыком `Print`, пока пользователь не выберет из списка объект, который требуется распечатать. Настройка свойства `Enabled` командной кнопки с именем `PrintButton` выполняется следующим образом:

```
PrintButton.Enabled = False
```

В большинстве случаев выбор момента настройки свойств является делом вашего вкуса. Я предпочитаю выполнять эту операцию внутри кода, так как это делает код более полным, а следовательно, и более читаемым. Кроме того, упрощается процесс редактирования свойств. Впрочем, ряд фундаментальных свойств, к которым относятся и свойства `Name` и `Caption`, лучше задавать в режиме конструирования.

Кнопка активизации

Поместите в форму командную кнопку, как показано на рис. 14.5. Для этого нужно щелкнуть на кнопке `CommandButton` панели инструментов `Toolbox` и опять же нарисовать в форме прямоугольную область. В окне `Properties` настройте свойства, перечисленные в табл. 14.2.

Таблица 14.2. Свойство кнопки активизации

Свойство	Значение
Name	CmdActivate
Accelerator	A
Caption	Активизировать
TabIndex	1

Кнопка отмены

Поместите в форму еще одну командную кнопку, расположенную под первой. В окне `Properties` настройте свойства, перечисленные в табл. 14.3.

Таблица 14.3. Свойства кнопки отмены

Свойство	Значение
Name	CmdCancel
Accelerator	C
Caption	Отмена
TabIndex	2
Cancel	True

Создание кода для объекта UserForm

Теперь пришло время связать с каждым элементом управления некий код.

Код для кнопки отмены

Дважды щелкните на кнопке Отмена, чтобы вызвать оболочку кода для события Click. Введите строку:

```
Unload Me
```

В результате оболочка кода будет выглядеть следующим образом:

```
Private Sub cmdCancel_Click()  
    Unload Me  
End Sub.
```

а щелчок на кнопке Отмена будет приводить к выгрузке формы.

Процедура ActivateSelectedSheet

Теперь пришло время создать процедуру, которая будет активизировать выбранный лист. Эта процедура должна выполняться в следующих ситуациях:

- выделение имени листа в списке и нажатие кнопки Активизация или использование комбинации клавиш Alt + A.;
- двойной щелчок на имени листа в списке;
- выделение имени листа в списке и нажатие клавиши Enter.

Так как код будет использоваться в трех различных ситуациях, избежать его повторения можно, поместив процедуру, показанную в листинге 14.1, в раздел General объекта UserForm.

Листинг 14.1. Процедура ActivateSelectedSheet

```
Sub ActivateSelectedSheet()  
    If lstSheets.ListIndex > -1 Then  
        Sheets(lstSheets.List(lstSheets.ListIndex)).Activate  
    End If  
    Unload Me  
End Sub
```

Этот код демонстрирует свойства поля со списком. Свойство ListIndex возвращает номер (нумерация начинается с нуля) выделенного в текущий момент элемента списка. Следовательно, строка

```
If lstSheets.ListIndex > -1 Then
```

проверяет, выделен ли какой-нибудь из элементов.

Код

```
lstSheets.List(i)
```

возвращает элемент с номером i поля со списком в виде строки. Соответственно:

```
lstSheets.List(lstSheets.ListIndex). -
```

это элемент, выделенный в данный момент, то есть имя листа. Ну и наконец, код

```
Sheets(lstSheets.List(lstSheets.ListIndex)).Activate
```

активизирует этот лист с помощью метода Activate. Более подробно этот метод будет рассмотрен в главе 18. На данный момент вам требуется всего лишь понимать, что код

```
Sheets("MySheet").Activate
```

активизирует лист с именем MySheet.

Последним выполняется событие cmdActivate_Click, которое выгружает не нужную в дальнейшем форму.

Кнопка активизации

Чтобы связать код с кнопкой Активизировать, выберите вариант cmdActivate в раскрываемся списке Objects, расположенном слева над окном кода, а в находящемся справа раскрываемся списке Procedures выберите вариант Click. Теперь можно включить в оболочку кода для события Click кнопки cmdActivate строку:

```
ActivateSelectedSheet
```

В итоге код будет выглядеть следующим образом:

```
Private Sub cmdActivate_Click()  
    ActivateSelectedSheet  
End Sub
```

Код для двойного щелчка кнопки мыши

Как уже упоминалось, процедура ActivateSelectedSheet должна выполняться, в том числе и при двойном щелчке пользователя на имени листа в списке. То есть при двойном щелчке на любом из элементов поля со списком должно вызываться событие DblClick. Выберите в раскрываемся списке Objects вариант lstSheets, а в раскрываемся списке Procedures — вариант DblClick. Затем введите в появившуюся оболочку следующий код:

```
Private Sub lstSheets_DblClick(ByVal Cancel As _  
    MSForms.ReturnBoolean)  
    ActivateSelectedSheet  
End Sub
```

Код для нажатия клавиши Enter

Процедура ActivateSelectedSheet должна выполняться и при нажатии клавиши Enter после выделения имени листа в поле со списком. То есть при нажатии клавиши должно вызываться событие KeyDown. Выберите это событие в раскрываемся списке Procedures и добавьте код, показанный в листинге 14.2.

Листинг 14.2. Процедура для события KeyDown объекта lstSheets

```
Private Sub lstSheets_KeyDown(ByVal KeyCode As _  
    MSForms.ReturnInteger, ByVal Shift As Integer)  
    If KeyCode = vbKeyReturn Then ActivateSelectedSheet  
End Sub
```

В данном случае требуется добавить код, определяющий, была ли нажата клавиша Enter. К счастью, Excel заполняет параметр KeyCode события KeyDown кодом для клавиши, нажатие на которую приводит к вызову этого события. Список кодов для различных клавиш можно найти в справочной документации VBA.

Код для поля со списком lstSheets

Теперь нужно сделать так, чтобы в поле lstSheets появился список всех листов текущей книги. Это должно происходить автоматически, поэтому соответствующий код следует связать с событием Initialize объекта UserForm. Это событие выполняется в начале загрузки формы до момента, когда она становится видимой. Как следует из названия события, оно предназначено для инициализации различных свойств форм и их элементов управления.

Выберите в раскрывающемся списке Objects вариант UserForm, а в раскрывающемся списке Procedures — вариант Initialize и введите в появившуюся оболочку код, показанный в листинге 14.3.

Листинг 14.3. Процедура события Initialize объекта UserForm

```
Private Sub UserForm_Initialize()  
    ' Заполняем текстовое поле lstSheets списком листов книги  
    Dim cSheets As Integer  
    Dim i As Integer  
    cSheets = Sheets.Count  
    lstSheets.Clear  
    For i = 1 To cSheets  
        lstSheets.AddItem Sheets(i).Name  
    Next  
End Sub
```

Сначала этот код определяет общее число листов в активной книге, затем очищает поле со списком от предыдущей информации. После этого с помощью цикла For в поле добавляются новые имена листов. Эта операция выполняется с помощью метода AddItem элементов управления поля со списком. Имя листа определяется его свойством Name.

Проверка работы служебной программы

Теперь, когда все куски кода на месте, пришло время сохранить книгу SRXUtils в виде надстройки и активизировать через меню Tools. Если надстройка с этим именем загружена в данный момент, то ее нужно выгрузить, иначе Excel откажется сохранять файл с расширением xla. Выберите в меню Tools редактора VBA команду Macros и запустите новую надстройку.

Элементы управления ActiveX на листе

Элементы управления ActiveX можно размещать на листах Excel, подобно стандартным элементам управления. Главное при этом не перегрузить лист элементами управления, которые могут быть помещены в объект UserForm.

Но в данном случае существуют некоторые обстоятельства, о которых требуется помнить. В частности каждый элемент управления ActiveX на *листе* представляется объектом OLEObject объектной модели Excel, который, в свою очередь, может включать в себя встроенные объекты OLE. К OLE-объектам относится, например, вставленное на лист растровое изображение.

Объект Worksheet имеет свойство OLEObjects, которое возвращает семейство OLEObjects, включающее все расположенные на листе OLE-объекты. Соответственно, код для семейства OLEObjects активного листа будет выглядеть так:

```
ActiveSheet.OLEObjects
```

Так как в это семейство входят встроенные OLE-объекты, нельзя сказать наверняка, что код:

```
ActiveSheet.OLEObjects(1)
```


описывает именно элемент управления. Поэтому при создании на листе элемента управления имеет смысл сразу присваивать ему имя и впоследствии обращаться уже по имени, а не по номеру. Например:

```
ActiveSheet.OLEObjects("MyButton")
```

Ссылка на расположенный на листе элемент управления

Как было сказано выше, Excel позволяет ссылаться на расположенные на листе элементы управления ActiveX без ссылки на семейство OLEObjects. Например, размещенной на листе командной кнопке автоматически присваивается системное имя CommandButton1. Вот две строки кода, каждая из которых делает высоту этой кнопки равной 20 точкам:

```
ActiveSheet.OLEObjects("CommandButton1").Height = 20
ActiveSheet.CommandButton1.Height = 20
```

К сожалению, свойства и методы, к которым можно получить доступ подобным образом, являются свойствами и методами объекта OLEObject, а не самого элемента управления. Эти свойства перечислены в табл. 14.4.

Таблица 14.4. Свойства и методы объекта OLEObject

AltHTML	Enabled	PrintObject
Activate	Height	ProgId
Application	Index	Select
AutoLoad	Interior	SendToBack
AutoUpdate	Left	Shadow
Border	LinkedCell	ShapeRange
BottomRightCell	ListFillRange	SourceName
BringToFront	Locked	Top
Copy	Name	TopLeftCell
CopyPicture	Object	Update
Creator	OLEType	Verb
Cut	OnAction	Visible
Delete	Parent	Width
Duplicate	Placement	Zorder

Соответственно, при задании свойства Height командной кнопки для определения свойства Caption этой же кнопки мы не можем использовать код:

```
ActiveSheet.OLEObjects("CommandButton1").Caption = "ClickMe"
```

Это приведет к появлению сообщения об ошибке.

Для получения доступа к свойствам и методам элементов управления надо использовать свойство Object объекта OLEObject, которое возвращает элемент управления. Вот правильные варианты задания свойства Caption:

```
ActiveSheet.OLEObjects("CommandButton1").Object.Caption = "ClickMe"
ActiveSheet.CommandButton1.Object.Caption = "ClickMe"
```

Элементы управления ActiveX, встроенные в лист Excel, имеют следующие свойства, отличные от стандартных:

- `BottomRightCell` — возвращает объект `Range`, который представляет собой ячейку, расположенную под нижним правым углом элемента управления.
- `LinkedCell` — возвращает диапазон ячеек листа, связанный со значением элемента управления. Соответственно, элемент управления будет принимать значение, введенное в связанную ячейку, и наоборот.
- `ListFillRange` — возвращает диапазон ячеек, значения которых использовались для заполнения поля со списком.
- `Placement` — возвращает или устанавливает способ, которым элемент управления соединен с расположенной под ним ячейкой. Возможные значения константы `xlPlacement`: `xlMoveAndSize`, `xlMove` и `xlFreeFloating`.
- `PrintObject` — если это свойство имеет значение `True`, при печати листа печатается и соответствующий элемент управления.
- `TopLeftCell` — возвращает объект `Range`, который представляет собой ячейку, расположенную над верхним левым углом элемента управления.
- `ZOrder` — возвращает `ZOrder` положение элемента управления.

Имейте в виду, что в табл. 14.4 перечислены свойства, которые относятся к OLE-объектам, являющимся контейнерами для элементов управления, а соответственно, к связи между листом и элементом. Например, код:

```
ActiveSheet.CommandButton1.TopLeftCell.Address
```

возвращает адрес левой верхней ячейки листа, расположенной под элементом управления, точнее, в его OLE-контейнере.

В качестве другого примера рассмотрим код, локализирующий в диапазоне ячеек, расположенных под командной кнопкой, левую верхнюю ячейку и затем делающий ее левой верхней ячейкой окна программы.

```
Dim rng As Range
Set rng = ActiveSheet.CommandButton1.TopLeftCell
With ActiveWindow
    .ScrollRow = rng.Row
    .ScrollColumn = rng.Column
End With
```

При активизации элементов управления ActiveX становятся недоступными определенные свойства и методы ряда объектов. Например, при активной кнопке невозможно использовать метод `Sort` объекта `Range`. То есть после щелчка на командной кнопке данный код работать не будет:

```
Private Sub CommandButton1_Click
    Range("A:A").Sort Key1:=Range("A:A")
End Sub
```

Это один из недостатков размещения элементов управления непосредственно на листе.

Разумеется, существует способ обойти эту проблему. Достаточно перед вызовом метода `Sort` активизировать другой объект. Например, так:

```
Private Sub CommandButton1_Click
    Range("A:A").Activate
    Range("A:A").Sort Key1:=Range("A:A")
    CommandButton1.Activate ' Optional
End Sub
```


Также стоит упомянуть о том, что при сохранении книги Excel 97/2000 в формате файлов Excel 5.0/95 информация об элементах управления ActiveX будет потеряна.

Программный способ добавления элементов управления на лист

Чтобы программным способом добавить на лист элемент управления ActiveX, используется метод Add из семейства OLEObjects. Вот синтаксис этого метода:

```
OLEObjectCollection.Add(ClassType, FileName, Link, DisplayAsIcon, _  
    IconFileName, IconIndex, IconLabel, Left, Top, Width, Height)
```

Параметр ClassType является так называемым программным идентификатором (ProgID) элемента управления. В табл. 14.5 перечислены программные идентификаторы для различных элементов управления.

Таблица 14.5. Программные идентификаторы для элементов управления ActiveX

Элемент управления	ProgID
Control	ProgID
CheckBox	Forms.CheckBox.1
ComboBox	Forms.ComboBox.1
CommandButton	Forms.CommandButton.1
Frame	Forms.Frame.1
Image	Forms.Image.1
Label	Forms.Label.1
ListBox	Forms.ListBox.1
MultiPage	Forms.MultiPage.1
OptionButton	Forms.OptionButton.1
ScrollBar	Forms.ScrollBar.1
SpinButton	Forms.SpinButton.1
TabStrip	Forms.TabStrip.1
TextBox	Forms.TextBox.1
ToggleButton	Forms.ToggleButton.1

Из остальных параметров к процессу создания элемента управления ActiveX относятся только параметры Left, Top, Width и Height, указывающие расположение (относительно левого верхнего угла листа, то есть ячейки A1) и размер элемента. Все остальные параметры нужно пропустить.

Например, код:

```
ActiveSheet.OLEObjects.Add ClassType:="Forms.Textbox.1", _  
    Left:=72, Top:=72, Height:=20, Width:=100
```

помещает новое текстовое поле на расстояние приблизительно одного дюйма от верхнего и левого краев активного листа.

15 Объектная модель Excel

Объектная модель Excel является одной из самых громоздких объектных моделей в арсенале Office. Она содержит почти 200 объектов и свыше 5000 свойств и методов. Однако, как уже упоминалось, многие из них предназначены исключительно для совместимости с более ранними версиями Excel. Если их игнорировать, число объектов снизится до 140, а число свойств и методов — до 3000. В результате объектная модель Excel будет уступать по размеру объектной модели Word.

Объекты и их свойства и методы, которые предназначены исключительно для совместимости с более ранними версиями, в данной книге рассматриваться не будут. Но следует, по крайней мере, знать об их существовании.

Подробный рассказ обо всех объектах и их свойствах и методах выходит за рамки данной книги, поэтому познакомимся только с главной частью объектной модели Excel. Остальные сведения, при необходимости, вы сможете получить самостоятельно.

Итак, начнем с обзора объектной модели Excel.

Обзор объектной модели Excel

Для начала сравним объектную модель Excel с объектными моделями других приложений Office 97. Для более поздних версий Office числа будут больше.

Таблица 15.1. Статистика по объектным моделям различных приложений Office 97

Приложение	Объекты	Свойства	Методы	Перечисления	Константы
Access 8	51	1596	532	31	485
Binder 8	4	37	15	4	11
DAO 3.5	37	235	174	26	185
Excel 8	192	3245	1716	152	1266
Forms 2	64	588	352	42	191
Graph 8	44	1120	234	58	447
Office 97	40	615	209	78	801
Outlook 8	42	1568	534	34	154
PowerPoint 8	110	1197	322	53	370
Word 8	188	2300	837	192	1969

В табл. 15.2 перечислены все явные объекты объектной модели Excel XP. В скобках указано число дочерних объектов.

Таблица 15.2. Объекты Excel 10 с указанием количества дочерних объектов

AddIn (1)	FillFormat (1)	Protection (1)
AddIns (2)	Filter (1)	PublishObject (1)
Adjustments (0)	Filters (2)	PublishObjects (2)
AllowEditRange (2)	Floor (4)	QueryTable (3)
AllowEditRanges (1)	Font (1)	QueryTables (2)
Application (32)	FormatCondition (4)	Range (22)
Areas (2)	FormatConditions (2)	RecentFile (2)
AutoCorrect (1)	FreeformBuilder (2)	RecentFiles (2)
AutoFilter (3)	Graphic (1)	RoutingSlip (1)
AutoRecover (1)	Gridlines (2)	RTD (0)
Axes (2)	GroupShapes (3)	Scenario (2)
Axis (6)	HiLoLines (2)	Scenarios (2)
AxisTitle (6)	HPageBreak (3)	Series (6)
Border (1)	HPageBreaks (2)	SeriesCollection (2)
Borders (2)	Hyperlink (3)	SeriesLines (2)
CalculatedFields (2)	Hyperlinks (2)	ShadowFormat (1)
CalculatedItems (2)	Interior (1)	Shape (22)
CalculatedMember (1)	IRtdServer (0)	ShapeNode (0)
CalculatedMembers (2)	IRTDUpdateEvent (0)	ShapeNodes (1)
CalloutFormat (0)	LeaderLines (2)	ShapeRange (17)
CellFormat (4)	Legend (5)	Shapes (4)
Characters (2)	LegendEntries (2)	Sheets (3)
Chart (16)	LegendEntry (3)	SmartTag (4)
ChartArea (5)	LegendKey (4)	SmartTagAction (1)
ChartColorFormat (1)	LineFormat (1)	SmartTagActions (2)
ChartFillFormat (2)	LinkFormat (1)	SmartTagOptions (1)
ChartGroup (7)	Mailer (1)	SmartTagRecognizer (1)
ChartGroups (2)	Name (2)	SmartTagRecognizers (2)
ChartObject (6)	Names (2)	SmartTags (2)
ChartObjects (6)	ODBCError (1)	SoundNote (1)
Charts (4)	ODBCErrors (2)	Speech (0)
ChartTitle (6)	OLEDLError (1)	SpellingOptions (0)
ColorFormat (0)	OLEDDBErrors (2)	Style (4)
Comment (3)	OLEFormat (1)	Styles (2)
Comments (2)	OLEObject (5)	Tab (1)
ConnectorFormat (2)	OLEObjects (6)	TextEffectFormat (0)
ControlFormat (1)	Outline (1)	TextFrame (2)
Corners (1)	PageSetup (2)	ThreeDFormat (1)
CubeField (3)	Pane (2)	TickLabels (2)
CubeFields (2)	Panes (2)	TreeviewController (1)
CustomProperties (2)	Parameter (2)	Trendline (3)

CustomProperty (1)	Parameters (2)	Trendlines (2)
CustomView (1)	Phonetic (2)	UpBars (4)
CustomViews (2)	Phonetics (2)	UsedObjects (1)
DataLabel (6)	PictureFormat (0)	UserAccess (0)
DataLabels (6)	PivotCache (2)	UserAccessList (1)
DataTable (3)	PivotCaches (2)	Validation (1)
DefaultWebOptions (1)	PivotCell (6)	VPageBreak (3)
Diagram (2)	PivotField (5)	VPageBreaks (2)
DiagramNode (3)	PivotFields (2)	Walls (4)
DiagramNodeChildren (1)	PivotFormula (1)	Watch (1)
DiagramNodes (1)	PivotFormulas (2)	Watches (2)
Dialog (1)	PivotItem (4)	WebOptions (1)
Dialogs (2)	PivotItemList (2)	Window (7)
DisplayUnitLabel (6)	PivotItems (2)	Windows (2)
DownBars (4)	PivotLayout (4)	Workbook (14)
DropLines (2)	PivotTable (8)	Workbooks (2)
Error (1)	PivotTables (2)	Worksheet (17)
ErrorBars (2)	PlotArea (4)	WorksheetFunction (1)
ErrorCheckingOptions (1)	Point (5)	Worksheets (3)
Errors (2)	Points (2)	

В табл. 15.3 перечислены объекты Excel, имеющие по крайней мере пять дочерних объектов. При сравнении табл. 15.2 и 15.3 нетрудно заметить, что количество дочерних объектов у большинства объектов Excel меньше пяти.

Таблица 15.3. Объекты Excel 10 с пятью и более дочерними объектами

Application (32)	DataLabel (6)	Point (5)
Axis (6)	DataLabels (6)	Range (22)
AxisTitle (6)	DisplayUnitLabel (6)	Series (6)
Chart (16)	Legend (5)	Shape (22)
ChartArea (5)	OLEObject (5)	ShapeRange (17)
ChartGroup (7)	OLEObjects (6)	Window (7)
ChartObject (6)	PivotCell (6)	Workbook (14)
ChartObjects (6)	PivotField (5)	Worksheet (17)
ChartTitle (6)	PivotTable (8)	

Список объектов Excel 10 с двузначным числом дочерних объектов:

- Application (32)
- Shape (22)
- Range (22)
- Worksheet (17)
- ShapeRange (17)
- Chart (16)
- Workbook (14)

То есть мощная иерархия объектной модели Excel в основном сконцентрирована на семи объектах, которые в основном и будут рассматриваться в последующих частях данной книги.

Перечисления Excel

Интересно также взглянуть на список перечислений. Их имена, за единственным исключением (перечисление Constants), начинаются с приставки Xl. Перечисления для версий Excel8/9/10 находятся в табл. 15.4–15.6. В скобках указано количество констант, входящих в каждое из них. Обратите внимание: объектная модель включает в себя ряд достаточно больших перечислений. Вот список перечислений по меньшей мере с 20 константами:

- XlBuiltInDialog (241)
- Constants (167)
- XlChartType (73)
- XlApplicationInternational (45)
- XlFileFormat (43)
- XlRangeAutoFormat (43)
- XlPaperSize (42)
- XlClipboardFormat (33)
- XlChartItem (32)
- XlPivotFormatType (22)
- XlParameterDataType (21)
- XlPattern (20)

Таблица 15.4. Перечисления Excel 8 с указанием количества констант

Constants (163)	XlEnableSelection (3)	XlPivotFieldOrientation (5)
XlApplicationInternational (45)	XlEndStyleCap (2)	XlPivotTableSourceType (4)
XlApplyNamesOrder (2)	XlErrorBarDirection (2)	XlPlacement (3)
XlArrangeStyle (4)	XlErrorBarInclude (4)	XlPlatform (3)
XlArrowHeadLength (3)	XlErrorBarType (5)	XlPrintLocation (3)
XlArrowHeadStyle (5)	XlFileAccess (2)	XlPriority (3)
XlArrowHeadWidth (3)	XlFileFormat (39)	XlPTSelectionMode (6)
XlAutoFillType (11)	XlFillWith (3)	XlRangeAutoFormat (21)
XlAutoFilterOperator (6)	XlFilterAction (2)	XlReferenceStyle (2)
XlAxisCrosses (4)	XlFindLookIn (3)	XlReferenceType (4)
XlAxisGroup (2)	XlFormatConditionOperator (8)	XlRoutingSlipDelivery (2)
XlAxisType (3)	XlFormatConditionType (2)	XlRoutingSlipStatus (3)
XlBackground (3)	XlFormControl (10)	XlRowCol (2)
XlBarShape (6)	XlFormulaLabel (4)	XlRunAutoMacro (4)
XlBordersIndex (8)	XlHAlign (8)	XlSaveAction (2)
XlBorderWeight (4)	XlHighlightChangesTime (3)	XlSaveAsAccessMode (3)

XIBuiltInDialog (221)	XIIMEMode (11)	XISaveConflictResolution (3)
XICalculation (3)	XIInsertShiftDirection (2)	XIScaleType (2)
XICategoryType (3)	XILegendPosition (5)	XISearchDirection (2)
XICellInsertionMode (3)	XILineStyle (8)	XISearchOrder (2)
XICellType (10)	XILink (4)	XISheetType (5)
XIChartGallery (3)	XILinkInfo (2)	XISheetVisibility (3)
XIChartItem (29)	XILinkInfoType (3)	XISizeRepresents (2)
XIChartLocation (3)	XILinkType (2)	XISortMethod (2)
XIChartPicturePlacement (7)	XILocationInTable (9)	XISortMethodOld (2)
XIChartPictureType (3)	XILookAt (2)	XISortOrder (2)
XIChartSplitType (4)	XIMailSystem (3)	XISortOrientation (2)
XIChartType (73)	XIMarkerStyle (12)	XISortType (2)
XIClipboardFormat (33)	XIMouseButton (3)	XISpecialCellsValue (4)
XIColorIndex (2)	XIMousePointer (4)	XISubscribeToFormat (2)
XICommandUnderlines (3)	XIMSAApplication (7)	XISummaryColumn (2)
XICommentDisplayMode (3)	XIObjectSize (3)	XISummaryReportType (2)
XIConsolidationFunction (11)	XIOLEType (3)	XISummaryRow (2)
XICopyPictureFormat (2)	XIOLEVerb (2)	XITabPosition (2)
XICreator (1)	XIOrder (2)	XITextParsingType (2)
XICutCopyMode (2)	XIOrientation (4)	XITextQualifier (3)
XICVError (7)	XIPageBreak (2)	XITickLabelOrientation (5)
XIDataLabelPosition (11)	XIPageBreakExtent (2)	XITickLabelPosition (4)
XIDataLabelsType (6)	XIPageOrientation (2)	XITickMark (4)
XIDataSeriesDate (4)	XIPaperSize (42)	XITimeUnit (3)
XIDataSeriesType (4)	XIParameterDataType (20)	XIToolbarProtection (5)
XIDeleteShiftDirection (2)	XIParameterType (3)	XITrendlineType (6)
XIDirection (4)	XIPasteSpecialOperation (5)	XIUnderlineStyle (5)
XIDisplayBlanksAs (3)	XIPasteType (6)	XIVAlign (5)
XIDisplayShapes (3)	XIPattern (20)	XIWBATemplate (4)
XIDVAlertStyle (3)	XIPhoneticAlignment (4)	XIWindowState (3)
XIDVType (8)	XIPhoneticCharacterType (4)	XIWindowType (5)
XIEditionFormat (4)	XIPictureAppearance (2)	XIWindowView (2)
XIEditionOptionsOption (8)	XIPictureConvertorType (13)	XIXLMMacroType (3)
XIEditionType (2)	XIPivotFieldCalculation (9)	XIYesNoGuess (3)
XIEnableCancelKey (3)	XIPivotFieldDataType (3)	

Таблица 15.5. Дополнительные перечисления Excel 9.0 с указанием количества констант

XICmdType (4)	XIHtmlType (4)	XISourceType (7)
XIColumnDataType (10)	XILayoutFormType (2)	XISubtotalLocationType (2)
XICubeFieldType (2)	XIPivotFormatType (22)	XIWebFormatting (3)
XIDisplayUnit (9)	XIQueryType (6)	XIWebSelectionType (3)

Таблица 15.6. Дополнительные перечисления Excel 10 и число их констант

XIArabicModes (4)	XIImportDataAs (2)	XIRobustConnect (3)
XICalculatedMemberType (2)	XIInsertFormatOrigin (2)	XISearchWithin (2)

продолжение ↗

Таблица 15.6 (продолжение)

XICalculationInterruptKey (3)	XILinkStatus (11)	XISmartTagDisplayMode (3)
XICalculationState (3)	XIPivotCellType (10)	XISortDataOption (2)
XICorruptLoad (3)	XIPivotTableMissingItems (3)	XISpeakDirection (2)
XIDataLabelSeparator (1)	XIPivotTableVersionList (3)	XIUpdateLinks (3)
XIErrorChecks (7)	XIPrintErrors (4)	
XIHebrewModes (4)	XIRangeValueDataType (3)	

Браузер объектов

Фирма Microsoft разработала специальный инструмент для просмотра объектов, свойств, методов, событий и перечислений объектной модели. Он носит имя Object Browser и вызывается одноименной командой меню View редактора VBA или нажатием клавиши F2. Вид окна браузера показан на рис. 15.1.



Рис. 15.1. Окно Object Browser

В левом верхнем углу расположен раскрывающийся список Project/Library, который позволяет выбрать объектную модель для просмотра. На рис. 15.1 выбрана объектная модель Excel. Ниже располагается текстовое поле Search Text, предназначенное для поиска объектов. В большом окне слева находится список классов объектной модели. Каждому объекту и каждому перечислению соответствует свой класс. В правом окне показаны свойства, методы и события для объекта, выделенного в левом окне. В расположенное в нижней части окна текстовое поле выводится информация о выделенном элементе.

Браузер объектов — очень полезный инструмент, и я не сомневаюсь, что вы не зря потратите время на его изучение. Но, к сожалению, он дает плоское, одномерное представление об объектной модели. Именно поэтому я написал собственную версию браузера, позволяющую получить двумерный вид, и именно он был использован для создания ряда иллюстраций в данной книге.

16 **Объект Application**

Как упоминалось в предыдущей главе, большинство действий с объектной моделью Excel сосредоточено на шести объектах: Application, Chart, PivotTable, Range, Workbook и Worksheet. В дальнейшем будут рассмотрены следующие объекты вместе с некоторыми их потомками:

- Application;
- Chart;
- CommandBars;
- Dialogs;
- Global;
- Names;
- Range;
- Sheets;
- Window/Windows;
- Workbook/Workbooks;
- Worksheet;
- WorkSheetFunctions.

Перечисленные в списке объекты образуют ядро объектной модели Excel. Они позволяют выполнить большинство задач, связанных с программированием, а также легко получить в справочной системе Excel дополнительную информацию для менее тривиальных случаев.

Некоторые из этих объектов настолько сложны, что описанию каждого из них можно посвятить целую главу. Примером подобного объекта может служить Application, а также ряд его потомков.

На рис. 16.1 показано, что объект Application находится на вершине иерархии объектной модели и представляет собой собственно приложение Excel и все его дочерние объекты. Обратите внимание на значки перед названиями объектов. Небольшой квадрат указывает на семейство объектов, а овал — на обычный объект.

На рис. 16.2 показаны все дочерние объекты Application, в том числе и те из них, которые являются скрытыми (соответствующие значки перечеркнуты). Объекты, рядом с которыми стоит пометка Office 2.2, в действительности име-

ют отношение к объектной модели Office. Но они могут применяться в ряде случаев при программировании в Excel. В данном списке фигурирует объект, принадлежащий к объектной модели Visual Basic Extensibility. Он помечен знаком VBIDE 5.3.

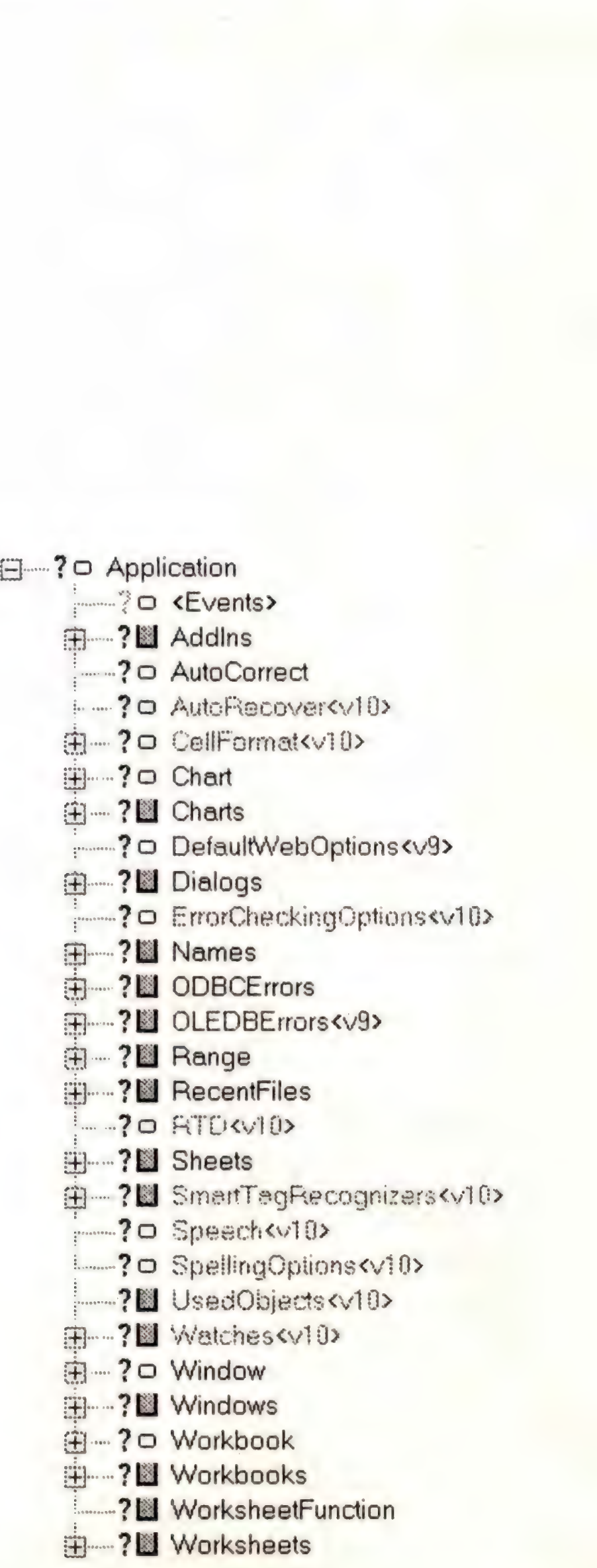


Рис. 16.1. Объект Application и его дочерние объекты. Знак <vX> означает, что объект впервые появился в версии X Excel

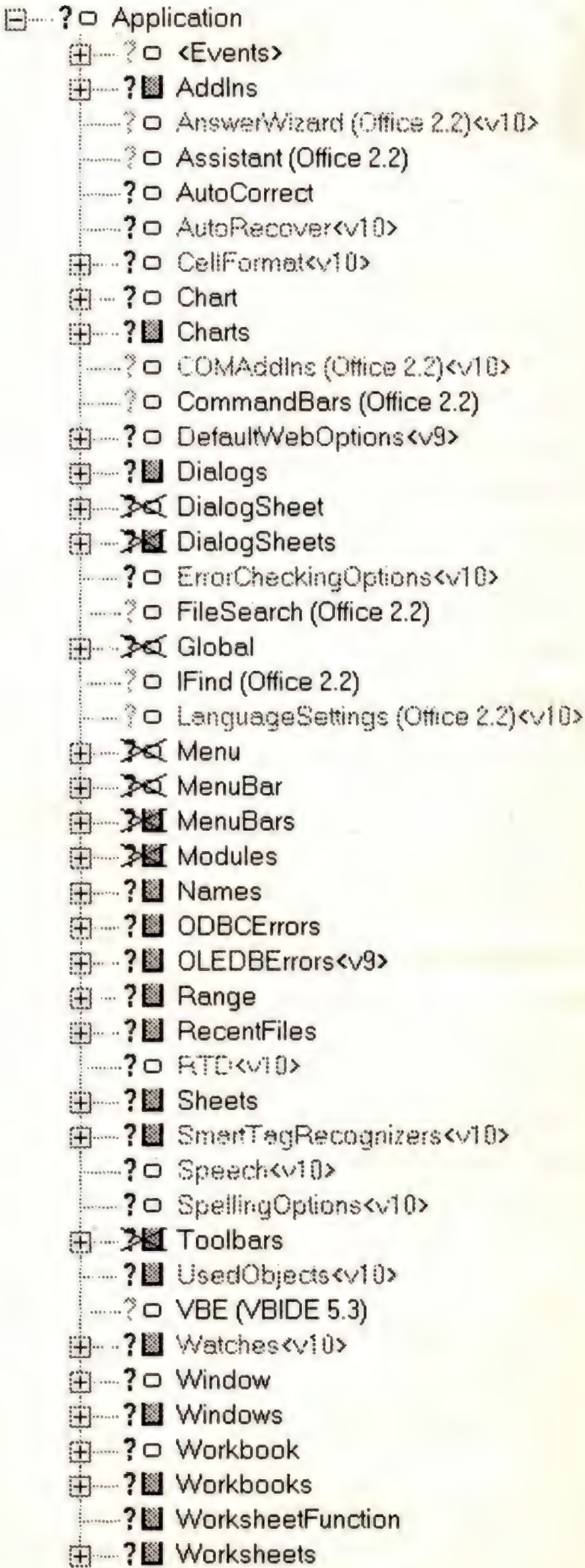


Рис. 16.2. Объект Application вместе со скрытыми дочерними объектами

Свойства и методы объекта Application

Объект Application имеет 286 свойств и методов, которые перечислены в табл. 16.1.

Таблица 16.1. Свойства и методы объекта Application

_Default	DisplayRecentFiles	OnDoubleClick
_Evaluate	DisplayScrollBars	OnEntry
_FindFile<v9>	DisplayStatusBar	OnKey
_Run2	DoubleClick	OnRepeat
_Wait<v9>	Dummy1	OnSheetActivate
_WSFunction	Dummy10	OnSheetDeactivate
ActivateMicrosoftApp	Dummy101<v9>	OnTime
ActiveCell	Dummy11	OnUndo
ActiveChart	Dummy12<v9>	OnWindow
ActiveDialog	Dummy13<v10>	OperatingSystem
ActiveMenuBar	Dummy14<v10>	OrganizationName
ActivePrinter	Dummy2	Parent
ActiveSheet	Dummy3	Path
ActiveWindow	Dummy4	PathSeparator
ActiveWorkbook	Dummy5	PivotTableSelection
AddChartAutoFormat	Dummy6	PreviousSelections
AddCustomList	Dummy7	ProductCode<v9>
AddIns	Dummy8	PromptForSummaryInfo
AlertBeforeOverwriting	Dummy9	Quit
AltStartupPath	EditDirectlyInCell	Range
AnswerWizard<v9>	EnableAnimations	Ready<v10>
Application	EnableAutoComplete	RecentFiles
AskToUpdateLinks	EnableCancelKey	RecordMacro
Assistant	EnableEvents	RecordRelative
AutoCorrect	EnableSound	ReferenceStyle
AutoFormatAsYouTypeReplaceHyperlinks<v10>	EnableTipWizard	RegisteredFunctions
AutomationSecurity<v10>	ErrorCheckingOptions<v10>	RegisterXLL
AutoPercentEntry<v9>	Evaluate	Repeat
AutoRecover<v10>	Excel4IntlMacroSheets	ReplaceFormat<v10>
Build	Excel4MacroSheets	ResetTipWizard
Calculate	ExecuteExcel4Macro	RollZoom
CalculateBeforeSave	ExtendList<v9>	Rows
CalculateFull<v9>	FeatureInstall<v9>	RTD<v10>
CalculateFullRebuild<v10>	FileConverters	Run
Calculation	FileDialog<v10>	Save
CalculationInterruptKey<v10>	FileFind	SaveWorkspace
CalculationState<v10>	FileSearch	ScreenUpdating
CalculationVersion<v9>	FindFile	Selection
Caller	FindFormat<v10>	SendKeys
CanPlaySounds	FixedDecimal	SetDefaultChart

CanRecordSounds	FixedDecimalPlaces	Sheets
Caption	GenerateGetPivotData<v10>	SheetsInNewWorkbook
CellDragAndDrop	GetCustomListContents	ShortcutMenus
Cells	GetCustomListNum	ShowChartTipNames
CentimetersToPoints	GetOpenFilename	ShowChartTipValues
Charts	GetPhonetic<v9>	ShowStartupDialog<v10>
CheckAbort<v10>	GetSaveAsFilename	ShowToolTips
CheckSpelling	Goto	ShowWindowsInTaskbar<v9>
ClipboardFormats	Height	SmartTagRecognizers<v10>
ColorButtons	Help	Speech<v10>
Columns	Hinstance<v10>	SpellingOptions<v10>
COMAddIns<v9>	Hwnd<v10>	StandardFont
CommandBars	IgnoreRemoteRequests	StandardFontSize
CommandUnderlines	InchesToPoints	StartupPath
ConstrainNumeric	InputBox	StatusBar
ControlCharacters	Interactive	TemplatesPath
ConvertFormula	International	ThisCell<v10>
CopyObjectsWithCells	Intersect	ThisWorkbook
Creator	Iteration	ThousandsSeparator<v10>
Cursor	LanguageSettings<v9>	Toolbars
CursorMovement	LargeButtons	Top
CustomListCount	Left	TransitionMenuKey
CutCopyMode	LibraryPath	TransitionMenuKeyAction
DataEntryMode	MacroOptions	TransitionNavigKeys
DDEAppReturnCode	MailLogoff	UILanguage
DDEExecute	MailLogon	Undo
DDEInitiate	MailSession	Union
DDEPoke	MailSystem	UsableHeight
DDERequest	MapPaperSize<v10>	UsableWidth
DDETerminate	MathCoproprocessorAvailable	UsedObjects<v10>
DecimalSeparator<v10>	MaxChange	UserControl
DefaultFilePath	MaxIterations	UserLibraryPath<v9>
DefaultSaveFormat	MemoryFree	UserName
DefaultSheetDirection	MemoryTotal	UseSystemSeparators<v10>
DefaultWebOptions<v9>	MemoryUsed	Value
DeleteChartAutoFormat	MenuBars	VBE
DeleteCustomList	Modules	Version
Dialogs	MouseAvailable	Visible
DialogSheets	MoveAfterReturn	Volatile
DisplayAlerts	MoveAfterReturnDirection	Wait
DisplayClipboardWindow	Name	Watches<v10>
DisplayCommentIndicator	Names	Width
DisplayExcel4Menus	NetworkTemplatesPath	Windows
DisplayFormulaBar	NewWorkbook<v10>	WindowsForPens
DisplayFullScreen	NextLetter	WindowState

продолжение ↗

Таблица 16.1 (продолжение)

DisplayFunctionToolTips<v10>	ODBCErrors	Workbooks
DisplayInfoWindow	ODBCTimeout	WorksheetFunction
DisplayInsertOptions<v10>	OleDbErrors<v9>	Worksheets
DisplayNoteIndicator	OnCalculate	
DisplayPasteOptions<v10>	OnData	

К сожалению, количество свойств и методов настолько велико, что даже для описания некоторой части из них потребуется отдельная книга. Поэтому мы сфокусируем внимание только на наиболее интересных и полезных из них. Таблица 16.1 позволит вам быстро найти свойства и методы, которые соответствуют определенным требованиям, и затем отыскать соответствующую информацию в справочной документации Excel.

Дополнительные свойства и методы объекта Application будут рассмотрены в последующих главах книги. Но чтобы придать обсуждению более структурированный характер, разобьем свойства и методы из табл. 16.1 на группы. Впрочем, это разбиение будет достаточно условным.

Свойства и методы, возвращающие дочерние объекты

Многие свойства и методы объекта Application предназначены для получения доступа к его дочерним объектам. Например, свойство Workbook просто возвращает семейство Workbooks, в которое входят все книги, открытые в текущий момент времени. Многие из этих свойств и методов будут обсуждаться чуть позже, а пока мы просто перечислим их.

Таблица 16.2. Свойства и методы, возвращающие дочерние объекты

Название свойства	Возвращаемый объект
ActiveCell	Range
ActiveChart	Chart
ActiveDialog	DialogSheet
ActiveMenuBar	MenuBar
ActiveWindow	Window
ActiveWorkbook	Workbook
AddIns	AddIns
Application	Application
AutoCorrect	AutoCorrect
AutoRecover	AutoRecover
Cells	Range
Charts	Sheets
Columns	Range
DefaultWebOptions	DefaultWebOptions
Dialogs	Dialogs
DialogSheets	Sheets
ErrorCheckingOptions	ErrorCheckingOptions

Название свойства	Возвращаемый объект
Excel4IntlMacroSheets	Sheets
Excel4MacroSheets	Sheets
FindFormat	CellFormat
Intersect	Range
MenuBar	MenuBar
Modules	Modules
Names	Names
NextLetter	Workbook
ODBCErrors	ODBCErrors
OleDbErrors	OleDbErrors
Parent	Application
Range	Range
RecentFiles	RecentFiles
ReplaceFormat	CellFormat
Rows	Range
RTD	RTD
Sheets	Sheets
ShortcutMenus	Menu
SmartTagRecognizers	SmartTagRecognizers
Speech	Speech
SpellingOptions	SpellingOptions
ThisCell	Range
ThisWorkbook	Workbook
Toolbars	Toolbars
Union	Range
UsedObjects	UsedObjects
Watches	Watches
Windows	Windows
Workbooks	Workbooks
WorksheetFunction	WorksheetFunction
Worksheets	Sheets

В таблице перечислено несколько свойств и методов, название которых начинается со слова **Active**. Это означает, что возвращается объект, который активен в данный момент. Например, свойство **ActiveSheet** возвращает активный лист с данными или с диаграммой. Обратите внимание на то, что объекта **Sheet** в реальности не существует, поскольку имеется разделение на листы с данными и листы с диаграммами.

Иногда имя свойства или метода совпадает с именем возвращаемого объекта. Например, свойство **AddIns** возвращает семейство **AddIns**, а свойство **Application** — объект **Application**. Впрочем, из этого правила существуют исключения:

- Свойство **ThisWorkbook** возвращает объект **Workbook**, содержащий выполняемый в данный момент код. Это свойство, в частности, позволяет определить положение книги (имя файла и маршрут доступа к нему) на компьютере пользователя. Для этого достаточно написать:

```
ThisWorkbook.FullName
```


- Некоторые свойства, например Cells, Columns и Rows, возвращают объект Range. Это связано с тем, что в объектной модели Excel отсутствуют объекты Cell, Column и Row. Ячейки, столбцы и строки являются объектами Range. То же самое можно сказать относительно объектной модели Word. В частности, отсутствуют объекты Character, Word и Sentence. Вместо них также фигурирует объект Range.

Свойства и методы, влияющие на отображение информации

Существуют свойства и методы, вызывающие отображение на экране определенных элементов:

- Если свойству DisplayAlert (чтение/запись, тип Boolean) присвоено значение True, приложение допускает отображение предупреждающих сообщений при запуске макросов. Например, от пользователя требуется подтверждение при удалении листа. Если вы не хотите, чтобы отображались подобные сообщения, присвойте данному свойству значение False.

```
Application.DisplayAlerts = False
```

По умолчанию это свойство имеет значение True.

- Свойство DisplayCommentIndicator (чтение/запись, тип Long) изменяет способ, с помощью которого отмечается наличие комментария в невыделенной ячейке. Оно может принимать значения констант из следующего перечисления:

```
Enum xlCommentDisplayMode
    xlCommentIndicatorOnly = -1      ' Показывает только индикатор
    xlNoIndicator = 0                ' Индикатор отсутствует
    xlCommentAndIndicator = 1        ' Отображает индикатор и
                                     комментарий
End Enum
```

Назначение свойству DisplayCommentIndicator значения xlCommentIndicatorOnly или CommentAndIndicator приводит к тому, что свойство DisplayNoteIndicator принимает значение True. Это значение изменяется на False в случае присвоения свойству DisplayCommentIndicator значения xlNoIndicator.

- Свойство DisplayFormulaBar (чтение/запись, тип Boolean) определяет, будет ли отображаться панель формул. По умолчанию, ему присвоено значение True.
- Свойство DisplayFullScreen (чтение/запись, тип Boolean) определяет, будет ли Excel работать в полноэкранном режиме. Имейте в виду, что этот режим не имеет отношения к эффекту разворачивания окна программы на полный экран.
- Если свойству DisplayNoteIndicator (чтение/запись, тип Boolean) присвоено значение True (задано по умолчанию), для ячеек, содержащих примечания, будут отображаться экранные подсказки. Кроме того, правый верхний угол таких ячеек будет отмечен маленькой точкой. Если это свойство имеет значение False, свойству DisplayCommentIndicator присваивается значение xlNoIndicator. В противном случае (значение True) последнее свойство имеет значение xlCommentIndicatorOnly.

Свойства и методы, влияющие на функции Excel

Ряд свойств и методов объекта Application разрешает или запрещает выполнение определенных функций Excel.

- Если свойство `AutoFormatAsYouTypeReplaceHyperlinks` (чтение/запись, тип `Boolean`) имеет значение `True`, это приводит к автоматическому форматированию гиперссылок. Если этой функции не требуется, достаточно присвоить свойству значение `False`.
- Свойство `EnableAnimations` (чтение/запись, тип `Boolean`) определяет возможность анимации процессов вставки и удаления. Если оно имеет значение `True`, происходит постепенное появление включенных строк и столбцов, а также постепенное исчезновение удаленных элементов. По умолчанию это свойство имеет значение `False`.
- Свойство `EnableAutoComplete` (чтение/запись, тип `Boolean`), по умолчанию имеющее значение `True`, определяет, будет ли включена функция автозаполнения.
- Свойство `EnableCancelKey` (чтение/запись, тип `Long`) определяет, каким образом приложение реагирует на нажатие комбинации клавиш `Ctrl + Break` или клавиши `Esc` во время выполнения программы. Оно может принимать значения констант из следующего перечисления:

```
Enum xlEnableCancelKey  
    xlDisabled = 0  
    xlInterrupt = 1  
    xlErrorHandler = 2  
End Enum
```

Вот значения этих констант:

- `xlDisabled`: нажатие клавиш игнорируется;
- `xlInterrupt`: выполняемая процедура прерывается и появляется диалоговое окно, в котором пользователь может как отладить, так и прервать процедуру (по умолчанию свойство `EnableCancelKey` имеет именно это значение);
- `xlErrorHandler`: нажатие клавиши приводит к вызову программы обработки ошибки с помощью оператора `On Error GoTo` (эта ошибка имеет код 18).

Имейте в виду, что это свойство следует использовать крайне осмотрительно. В частности, если присвоить ему значение `xlDisabled`, способа прервать бесконечный цикл не будет. Аналогично, если это свойство имеет значение `xlErrorHandler`, а программа обработки ошибок использует оператор `Resume`, возникнет состояние бесконечного выполнения цикла.

Именно поэтому после завершения выполнения кода свойству `EnableCancelKey` автоматически присваивается значение `xlInterrupt`.

- Свойство `EnableEvents` (чтение/запись, тип `Boolean`). Это свойство по умолчанию имеет значение `True`, и, соответственно, для объекта Application разрешены различные события. Подробно о событиях рассказывается в главе 11.
- Свойство `EnableSound` (чтение/запись, тип `Boolean`). Это свойство позволяет включать и выключать звуки при работе с пакетом Microsoft Office. По умолчанию оно имеет значение `False`.

Свойства и методы, связанные с событиями

С макросами могут связываться определенные события. Для этого существуют специальные события, не имеющие отношения к описанным в главе 11. Например, с помощью метода `OnKey` объекта `Application` можно сделать так, чтобы нажатие определенной клавиши сопровождалось звуком. Опишем два наиболее полезных события, связанных с выполнением определенных макросов.

Метод OnKey

Синтаксис метода `OnKey` выглядит следующим образом:

```
Application.OnKey(Key, Procedure).
```

где параметр *Key* представляет собой клавишу или комбинацию клавиш (записанные в виде строки), нажатие которых приводит к выполнению макроса, а *Procedure* — название макроса.

Назначив параметру *Key* соответствующую какой-либо операции комбинацию клавиш (например, `Ctrl + S` для сохранения), можно изменить обычное поведение приложения. Если параметру *Procedure* назначить пустую строку, приложение не будет реагировать обычным образом, соответственно, ничего не произойдет. Пропуск данного параметра возвращает комбинации клавиш ее обычное значение.

Для иллюстрации рассмотрим код, блокирующий работу комбинации клавиш `Ctrl + o`, нажатие которой в обычном режиме приводит к появлению диалогового окна `Open` (Открытие документа):

```
Application.OnKey "^o", ""
```

Следующая строка возвращает этой комбинации клавиш ее традиционные функции:

```
Application.OnKey "^o"
```

Аргументу *Key* могут соответствовать как одна клавиша, так и комбинации с клавишами `Alt`, `Ctrl` и `Shift`. Обычные буквенно-цифровые клавиши обозначаются с помощью кавычек, например `"a"`, `"A"`, `"1"`. В табл. 16.3 перечислены обозначения специальных клавиш. Например, клавиша `F2` обозначается `{F2}`, а клавиша `Enter` — `{ENTER}`, или `{~}`.

Таблица 16.3. Обозначения специальных клавиш для параметра *Key*

Клавиша	Обозначение
Backspace	{BACKSPACE}, или {BS}
Break	{BREAK}
Caps Lock	{CAPSLOCK}
Clear	{CLEAR}
Delete или Del	{DELETE}, или {DEL}
Стрелка вниз (v)	{DOWN}
End	{END}
Enter (цифровая клавиатура)	{ENTER}
Enter	{~}
Esc	{ESCAPE}, или {ESC}
Help	{HELP}

Клавиша	Обозначение
Home	{HOME}
Ins	{INSERT}
Стрелка влево (<)	{LEFT}
Num Lock	{NUMLOCK}
Page Down	{PGDN}
Page Up	{PGUP}
Return	{RETURN}
Стрелка вправо (>)	{RIGHT}
Scroll Lock	{SCROLLLOCK}
Tab	{TAB}
Стрелка вверх (^)	{UP}
F1 — F15	{F1} — {F15}

Для обозначения комбинаций с клавишами Shift, Ctrl и Alt используются следующие префиксы.

Shift	+ (знак плюс)
Ctrl	^ (знак вставки)
Alt	% (знак процентов)

Например, комбинация клавиш Alt + F2 записывается как “%{F2}”, а комбинация клавиш Ctrl + Shift + Enter — “^+{ENTER}”.

Если требуется, чтобы знаки +, ^, % и ~ не интерпретировались как специальные клавиши, их следует заключить в фигурные скобки. Например, для обозначения клавиши { параметру *Key* присваивается значение “{ }”.

Метод OnTime

Метод OnTime используется для запуска процедур в определенное время или через определенный промежуток времени. Вот его синтаксис:

```
Application.OnTime(EarliestTime, Procedure, LatestTime, Schedule)
```

Очевидно, что параметр *Procedure* представляет собой имя запускаемого макроса, а параметр *EarleistTime* — время запуска. Для установки времени применяется функция TimeValue. К примеру, следующий код приводит к запуску макроса test в модуле ThisWorkbook книги book1 в 15 часов 58 минут:

```
Application.OnTime TimeValue("3:58 PM")._
"d:\excel\book1.xls!ThisWorkbook.test"
```

Параметр *LatestTime* является необязательным и указывает окончание временного промежутка, в течение которого может быть осуществлен запуск процедуры. Для определения этого параметра можно также использовать процедуру TimeValue. Или же он вычисляется по следующей формуле: *EarleistTime* плюс дополнительное время. Например, указанный ниже код должен приводить к запуску макроса не позднее чем через 30 секунд после 15 часов 58 минут:

```
"d:\excel\book1.xls!ThisWorkbook.test"._
TimeValue("3:58 PM") + 30
```

Параметр *LatestTime* может быть полезен в ситуациях, когда приложение занято выполнением другой программы, и запуск макроса откладывается. Но если при

этом требуется, чтобы запуск макроса не происходил после определенного времени, достаточно указать значение параметра *LatestTime*.

Когда требуется отменить ранее запрограммированный запуск макроса, параметру *Schedule* присваивается значение *False*. В остальных случаях его можно пропустить, так как по умолчанию он имеет значение *True*.

Функция *Now* возвращает текущее время. Соответственно, если требуется запустить макрос через определенное время, достаточно присвоить параметру *EarleistTime* значение *Now + TimeValue(time)*.

Свойства и методы, связанные с вычислениями

Объект Application обладает рядом свойств и методов, имеющих отношение к вычислениям.

Метод Calculate

В зависимости от способа применения метода Calculate вызывается пересчет формул во всех открытых книгах, на определенных листах указанных книг или в определенном диапазоне ячеек листа.

Когда данный метод применяется к объекту Application в форме:

`Application.Calculate,`

происходит пересчет всех формул во всех открытых книгах. Впрочем, в данном случае можно написать просто:

`Calculate.`

так как этот метод является глобальным. Если же для определенного листа записать:

`Worksheets(1).Calculate,`

произойдет пересчет всех формул на этом листе. Когда требуется пересчитать формулы в некотором диапазоне ячеек, используется строка:

`Worksheets(1).Rows(2).Calculate`

Метод CalculateFullRebuild

Метод CalculateFullRebuild производит пересчет всех формул и восстановление всех зависимостей (формул, которые ссылаются на другие ячейки) во всех открытых книгах. Этот метод применим исключительно к объекту Application.

Свойство Calculation (чтение/запись, тип Long)

Свойство Calculation используется для настройки режима пересчета формул в Excel и может принимать значения констант из следующего перечисления:

```
Enum xlCalculation
    xlCalculationManual = -4135
    xlCalculationAutomatic = -4105
    xlCalculationSemiautomatic = 2
End Enum
```

По умолчанию это свойство имеет значение *xlCalculationAutomatic*. Смысл значения *Semiautomatic* в документации не объясняется, по крайней мере автору подобные объяснения на глаза не попались. Однако на вкладке Calculation (Вычисления) диалогового окна Options (Параметры) можно задать режим автоматических

вычислений везде, кроме таблиц. Именно этот случай описывает значение Semi-automatic.

Свойство CalculateBeforeSave (чтение/запись, тип Boolean)

Свойство CalculateBeforeSave имеет значение True, если пересчет формул в книгах произошел до сохранения последних на диске. Оно работает только в том случае, когда свойство Calculation имеет значение xlManual.

Метод CheckAbort

Метод CheckAbort останавливает процесс пересчета формул в Excel. Вот его синтаксис:

```
Application.CheckAbort(keepabortrange).
```

где аргумент *keepabortrange* представляет собой диапазон ячеек, в котором метод не действует, то есть пересчет формул все равно производится.

Свойства и методы, связанные с файлами

Перечислим методы и свойства, которые связаны с управлением файлами.

Свойство DefaultFilePath (чтение/запись, тип String)

Свойство DefaultFilePath возвращает или задает маршрут доступа, который по умолчанию используется приложением при открытии и сохранении файлов. Изменить эти настройки можно на вкладке General (Общие) диалогового окна Options (Параметры).

Свойство DefaultSaveFormat (чтение/запись, тип Long)

Свойство DefaultSaveFormat возвращает набор форматов, используемых при сохранении файлов. По умолчанию оно имеет значение xlWorkbookNormal, указывая на обычный формат книги для текущей версии Excel. Возможными значениями являются также константы XLFileFormat, перечисленные в табл. 16.4.

Таблица 16.4. Константы XLFileFormat

xlAddIn (18)	xlExcel5 (39)	xlWJ3 (40)
xlCSV (6)	xlExcel7 (39)	xlWJ3FJ3 (41)
xlCSVMac (22)	xlExcel9795 (43)	xlWK1 (5)
xlCSVMSDOS (24)	xlHtml (44)<v9>	xlWK1ALL (31)
xlCSVWindows (23)	xlIntlAddIn (26)	xlWK1FMT (30)
xlCurrentPlatformText (-4158)	xlIntlMacro (25)	xlWK3 (15)
xlDBF2 (7)	xlSYLK (2)	xlWK3FM3 (32)
xlDBF3 (8)	xlTemplate (17)	xlWK4 (38)
xlDBF4 (11)	xlTextMac (19)	xlWKS (4)
xlDIF (9)	xlTextMSDOS (21)	xlWorkbookNormal (-4143)
xlExcel2 (16)	xlTextPrinter (36)	xlWorks2FarEast (28)
xlExcel2FarEast (27)	xlTextWindows (20)	xlWQ1 (34)
xlExcel3 (29)	xlUnicodeText (42)<v9>	xlXMLSpreadsheet (46)<v10>
xlExcel4 (33)	xlWebArchive (45)<v10>	
xlExcel4Workbook (35)	xlWJ2WD1 (14)	

Свойство FileDialog

Свойство FileDialog программным образом открывает диалоговые окна, связанные с файлами, и возвращает объект FileDialog. В качестве параметра используется одна из следующих констант, указывающая на тип диалогового окна:

```
msoFileDialogFilePicker
msoFileDialogFolderPicker
msoFileDialogOpen
msoFileDialogSaveAs
```

Для иллюстрации рассмотрим код, позволяющий пользователю выбрать папку и узнать путь доступа к ней:

```
' Открыть диалоговое окно
With Application.FileDialog(msoFileDialogFolderPicker)
    .Show
    ' Показать путь доступа
    MsgBox .SelectedItems(1)
End With
```

Метод FindFile

Метод FindFile имеет следующий синтаксис:

```
Application.FindFile
```

Он вызывает появление диалогового окна Open (Открытие документа). Если пользователь открывает файл, метод возвращает значение True. Если же диалоговое окно было закрыто путем щелчка на кнопке Cancel (Отмена), метод возвращает значение False.

Метод GetOpenFilename

Метод GetOpenFilename приводит к появлению диалогового окна Open (Открытие документа) с указанным пользователем именем файла, но при этом сам файл не открывается. Вот его синтаксис:

```
Application.GetOpenFilename(FileFilter,
    FilterIndex, Title, ButtonText, MultiSelect)
```

Необязательный параметр *FileFilter* представляет собой строку, которая указывает, что будет содержаться в раскрывающемся списке File of Type (Тип файлов) диалогового окна Open (Открытие документа). Соответственно, первая часть также включает спецификацию фильтрации файлов по групповым символам. Поясним это на примерах.

Текстовые файлы:

```
Text Files (*.txt),*.txt
```

Файлы Lotus:

```
Lotus Files (*.wk?). *wk?
```

Файлы надстроек:

```
Add-In Files (*.xla),*.xla
```

Также допустимо использовать фильтрацию файлов с несколькими групповыми символами. Например:

```
Backup Files (*.xlk; *.bak), *.xlk; *.bak
```

Обратите внимание на наличие точки с запятой. При пропуске аргумента *FileFilter* по умолчанию следует:

```
All Files (*.*),*.*
```


Аргумент *FileFilter* может содержать несколько фильтров, разделенных запятыми. Например:

```
Debug.Print Application.GetOpenFilename( _  
    "Text Files (*.txt);*.txt;  
    Backup Files (*.xlk;*.bak);*.xlk;*.bak")
```

В этом случае необязательный параметр *FilterIndex* указывает, какой из фильтров появится в раскрывающемся списке File of Type (Тип файлов). К примеру, следующий код приведет к появлению в этом списке второго фильтра (для резервных копий файлов):

```
Debug.Print Application.GetOpenFilename( _  
    "Text Files (*.txt);*.txt;  
    Backup Files (*.xlk;*.bak);*.xlk;*.bak", 2)
```

Необязательный параметр *Title* определяет название диалогового окна. В указанном примере пропуск этого аргумента приводит к появлению диалогового окна Open (Открытие документа). Параметр *ButtonText* игнорируется в операционной системе Windows, но и используется в Macintosh.

Если необязательное свойство *MultiSelect* имеет значение True, появляется возможность выбора нескольких имен файлов, в то время как значение False допускает выбор только одного из них. По умолчанию это свойство имеет значение False. Для выбора нескольких файлов в диалоговом окне Open (Открытие документа) требуется удерживать нажатой клавишу Ctrl или Shift.

Рассматриваемый метод возвращает выделенное или введенное пользователем имя файла. При этом может быть указан путь доступа. Если параметр *MultiSelect* имеет значение True, возвращается массив имен файлов (даже если было выделено только одно имя). При этом количество выделенных пользователем файлов можно определить с помощью функции UBound, которая определяет верхнюю границу возвращенного массива:

```
NumFiles = UBound(Application.GetOpenFilename(MultiSelect:=True))
```

При закрытии диалогового окна пользователем метод возвращает значение False. Кроме того, этот метод может изменять текущую папку или даже диск.

Метод GetSaveAsFilename

Метод GetSaveAsFilename вызывает отображение диалогового окна Save As (Сохранение документа). Имя сохраняемого файла вводится пользователем. Вот синтаксис этого метода:

```
Application.GetSaveAsFilename(InitialFilename, _  
    FileFilter, FilterIndex, Title, ButtonText)
```

Необязательный параметр *InitialFilename* определяет имя файла, помещаемое в текстовое поле File name (Имя файла) диалогового окна Save As (Сохранение документа). При пропуске этого аргумента используется имя активной книги. Все остальные параметры и возвращаемые значения ничем не отличаются от тех, которые описаны в предыдущем разделе. Имейте в виду, что метод GetSaveAsFilename также может применяться для изменения текущей папки и даже диска.

Свойство RecentFiles (только для чтения)

Свойство RecentFiles возвращает семейство RecentFiles, включающее имена последних открывавшихся файлов. Этому семейству присущи две особенности. Во-пер-

вых, включает свойство `Maximum`, возвращающее или задающее максимальное число имен файлов, которые могут отображаться в списке меню `File` (Файл). Это целое число от 0 до 9 включительно. Соответственно, код

```
MsgBox Application.RecentFiles.Maximum
```

приводит к появлению информационного окна с текущим значением.

Во-вторых, существует возможность распечатки списка имен недавно открывавшихся файлов с помощью следующего кода:

```
Dim rf As RecentFile
For Each rf In Application.RecentFiles
    Debug.Print rf.Name
Next
```

Семейство `RecentFiles` включает объекты `RecentFile`, а не просто имена последних открывавшихся файлов, как можно было ожидать.

Метод `SaveWorkspace`

Метод `SaveWorkspace` сохраняет текущую рабочую область. Вот его синтаксис:

```
Application.SaveWorkspace(Filename)
```

Параметр *Filename* является необязательным и представляет собой имя `xlw`-файла.

Свойства и методы, влияющие на текущее состояние Excel

Перечислим свойства и методы, оказывающие влияние на состояние Excel:

- Если свойство `CopyObjectsWithCells` (чтение/запись, тип `Boolean`) имеет значение `True`, объекты (например, встроенные кнопки или формы) вырезаются, копируются, извлекаются и сортируются вместе с ячейками.
- Свойство `Cursor` (чтение/запись, тип `Long`) возвращает или определяет вид указателя курсора мыши; оно может принимать значения констант из следующего перечисления:

```
Enum XlMousePointer
    xlDefault = -4143
    xlNorthwestArrow = 1
    xlWait = 2
    xlIBeam = 3
End Enum
```

Если выполнение кода занимает некоторое время, желательно присвоить свойству `Cursor` значение `xlWait`. Разумеется, после завершения выполнения процедуры указатель курсора мыши должен быть возвращен в исходное состояние. Это проще всего сделать путем сохранения исходного значения свойства `Cursor`, что позволяет вернуться к первоначальным настройкам.

- Свойство `CutCopyMode` (чтение/запись, тип `Long`), возвращающее или устанавливающее состояние режимов `Cut` или `Copy`, может иметь значение `True` или `False`. Они равноценны по своему значению для платформы PC и различаются для Macintosh. В данном случае речь идет о способах отмены режимов `Cut` и `Copy` и удаления рамки, окружающей копируемую или вырезаемую область.

Если свойство CutCopyMode возвращает значение False, значит, в Excel в данный момент не включены ни режим Cut, ни режим Copy. Также данное свойство может возвращать значения из следующего перечисления:

```
Enum XlCutCopyMode
    xlCopy = 1          ' Режим Copy
    xlCut = 2           ' Режим Cut
End Enum
```

- Свойство DataEntryMode (чтение/запись, тип Long) возвращает или устанавливает режим ввода данных. В этом режиме данные могут быть введены только в ячейки выделенного в текущий момент диапазона. Это свойство способно принимать следующие значения:
 - xlOn. Режим ввода данных включен.
 - xlOff. Режим ввода данных выключен.
 - xlStrict. Режим ввода данных включен и не выключается при нажатии клавиши Esc.
- Свойство EditDirectlyInCell (чтение/запись, тип Boolean) по умолчанию имеет значение True, что дает возможность редактирования в ячейках. В противном случае вносить изменения в содержимое ячеек можно только в строке формул.
- Если свойство FixedDecimal (чтение/запись, тип Boolean) имеет значение True, все вводимые числовые данные форматируются с учетом числа десятичных разрядов, заданного свойством FixedDecimalPlaces. По умолчанию свойство FixedDecimal имеет значение False. При этом значение свойства FixedDecimalPlaces игнорируется.
- Свойство FixedDecimalPlaces (чтение/запись, тип Long) возвращает или устанавливает число фиксированных десятичных разрядов, которые используются, если свойство FixedDecimal имеет значение True. Например, если свойство FixedDecimalPlaces имеет значение 3, введенное в ячейку число 100 будет отображено в виде 0.1.
- Если свойство Interactive (чтение/запись, тип Boolean) имеет значение False, Excel блокирует ввод данных с помощью клавиатуры и мыши. Для ввода данных в этом случае используются только вызванные программным способом диалоговые окна. Это делается в целях предотвращения ввода данных пользователем в процессе работы макроса. По умолчанию свойство Interactive имеет значение True.

Разумеется, использовать это свойство надлежит особо аккуратно. К примеру, если забыть вернуть заданное по умолчанию значение или если выполнение кода будет прервано, для возвращения к обычному режиму ввода данных пользователю потребуется перезагрузить Excel. При этом закрыть приложение можно с помощью комбинации клавиш Alt + F4, но несохраненные данные будут утеряны.

- Свойство MoveAfterReturn (чтение/запись, тип Boolean) по умолчанию имеет значение True. При этом активная ячейка перемещается после нажатия клавиши Enter. Направление этого перемещения указывается свойством MoveAfterReturnDirection. Если свойству MoveAfterReturn присвоить значение False, нажатие клавиши Enter не будет оказывать влияния на положение активной ячейки.
- Свойство MoveAfterReturnDirection (чтение/запись, тип Long) возвращает или задает направление, в котором будет перемещаться активная ячейка при нажа-

тии клавиши Enter, если свойство MoveAfterReturn имеет значение True. Оно может принимать значения констант из следующего перечисления:

```
Enum XlDirection  
    xlUp = -4162  
    xlToRight = -4161  
    xlToLeft = -4159  
    xlDown = -4121  
End Enum
```

- Свойство ReferenceStyle (чтение/запись, тип Long) возвращает или устанавливает стиль (A1 или R1C1) показа ссылок на ячейки и заголовков строк и столбцов. Оно принимает значения констант из следующего перечисления:

```
Enum XlReferenceStyle  
    xlR1C1 = -4150  
    xlA1 = 1  
End Enum
```

- Свойство ScreenUpdating (чтение/запись, тип Boolean) по умолчанию имеет значение True, что соответствует включенному режиму обновления экрана. Так как это может значительно замедлить выполнение отдельных процедур, в некоторых случаях бывает полезно присваивать этому свойству значение False.
- Свойство SheetsInNewWorkbook (чтение/запись, тип Long) возвращает или устанавливает количество листов, которые автоматически вставляются в новую книгу.
- Свойство ShowChartTipNames (чтение/запись, тип Boolean) по умолчанию имеет значение True, что делает видимыми подписи данных на диаграммах.
- Свойство ShowChartTipValues (чтение/запись, тип Boolean) по умолчанию имеет значение True и делает видимыми подписи значений на диаграммах.
- Свойство ShowToolTips (чтение/запись, тип Boolean) по умолчанию имеет значение True и вызывает появление всплывающих подсказок для панелей инструментов.
- Свойство StandardFont (чтение/запись, тип String) возвращает или устанавливает имя стандартного шрифта. Имейте в виду, что все изменения этого свойства вступают в силу только после перезагрузки Excel.
- Свойство StandardFontSize (чтение/запись, тип Long) возвращает или устанавливает размер стандартного шрифта (в точках). Изменения также вступают в силу только после перезагрузки Excel.
- Свойство StartupPath (только чтение, тип String) возвращает путь доступа к загрузочной папке, исключая последний разделительный знак.
- Свойство TemplatesPath (только чтение, тип String) возвращает маршрут доступа к папке с шаблонами.

Свойства и методы, приводящие к выполнению различных операций

Ряд свойств и методов объекта Application приводит к выполнению различных операций.

Метод ConvertFormula

Метод ConvertFormula используется для перехода между стилями A1 и R1C1 ссылок на ячейки в формулах. Он также осуществляет преобразование между относительными и абсолютными ссылками. Вот его синтаксис:

```
Application.ConvertFormula(Formula, FromReferenceStyle, ToReferenceStyle, _  
    ToAbsolute, RelativeTo)
```

Параметр *Formula* представляет собой строку с подлежащей преобразованию формулой. Это должна быть действительная формула, начинающаяся со знака «равно».

Параметр *FromReferenceStyle* принимает значения констант из следующего перечисления:

```
Enum XlReferenceStyle  
    xlR1C1 = -4150  
    xlA1 = 1  
End Enum
```

Необязательный параметр *ToReferenceStyle* описывает стиль ссылки, который будет использоваться при преобразовании формул. Это также одна из констант XlReferenceStyle. Пропуск этого аргумента приводит к тому, что стиль ссылок остается без изменений.

Необязательный параметр *ToAbsolute* указывает тип преобразованной ссылки и может принимать значения констант из следующего перечисления:

```
Enum XlReferenceType  
    xlAbsolute = 1  
    xlAbsRowRelColumn = 2  
    xlRelRowAbsColumn = 3  
    xlRelative = 4  
End Enum
```

Если этот аргумент пропущен, тип ссылок не меняется.

Ну и наконец, необязательный параметр *RelativeTo* является объектом Range, который содержит одну ячейку. Эта ячейка используется для определения относительных ссылок. То есть можно представить, что формула помещена в данную ячейку, и с учетом этого будут рассматриваться все относительные ссылки.

Для иллюстрации рассмотрим следующий код:

```
sFormula = "=D2"  
Debug.Print Application.ConvertFormula(sFormula, _  
    xlA1, xlR1C1, xlRelative, Range("C3"))  
Debug.Print Application.ConvertFormula(sFormula, _  
    xlA1, xlR1C1, xlRelRowAbsColumn, Range("C3"))
```

Вторая строка преобразовывает условное обозначение A1 в условное обозначение R1C1, предполагая, что формула находится в ячейке C3. Однако на выходе вы получите:

= R[-1]C[1].

так как ячейка D2 находится на один столбец вправо и на один ряд ниже ячейки C3. Третья строка кода преобразует условное обозначение A1 в условное обозначение R1C1, но использует абсолютную ссылку на столбец, поэтому в результате получим:

= R[-1]C4.

так как столбец номер 4 находится справа от столбца 3.

Метод Evaluate

Метод Evaluate преобразует имена в объекты или значения. Вот его синтаксис:

```
Application.Evaluate(Name)
```

Данный метод применим также к объектам Chart, DialogSheet и Worksheet.

Параметр *Name* представляет собой имя объекта. Он может принадлежать к одному из следующих типов:

- **Ссылка вида A1.** При этом ссылка считается абсолютной. Для иллюстрации рассмотрим код, каждая строка которого помещает слово «Маша» в ячейку A1:

```
Range("A1").Value = "Маша"
A1.Value = "Маша"
Evaluate("A1").Value = "Маша"
[A1].Value = "Маша"
```

Первая строка использует метод Range. Вторая строчка приведет к сообщению об ошибке, так как приложение рассматривает A1 как переменную, а не как ссылку на ячейку. В третьей строке используется метод Evaluate, который преобразует имя ячейки в объект Range. Четвертая строка является кратким вариантом записи третьей строки.

- **Диапазон.** Параметру *Name* может присваиваться значение диапазона, сформированного с помощью оператора диапазона (двоеточие), оператора пересечения (пробел) и оператора объединения (запятая). Для иллюстрации рассмотрим код:

```
Evaluate("B2:C4").Select
Evaluate("B2:C4, D5:F6").Select
Evaluate("B2:C4 B1:F2").Select
[B2:C4 B1:F2].Select
```

Первая строка выделяет диапазон B2:C4. Вторая строка выделяет результат объединения двух прямоугольных диапазонов B2:C4 и D5:F6. Третья строка приводит к выделению области пересечения двух прямоугольных диапазонов B2:C4 и B1:F2. Четвертая строка является кратким вариантом записи третьей.

- **Заданное имя.** К примеру, можно присвоить диапазону имя test, а затем выделить его с помощью следующей строки:

```
Evaluate("test").Select
```

К сожалению, используя подобный синтаксис, автор нередко получал противоречивые результаты. В некоторых случаях он работает абсолютно корректно, но по этому поводу ничего нельзя сказать наверняка. Кроме того, можно использовать названия формул. Например, следующий код показывает сумму значений в диапазоне B2:B5:

```
MsgBox Evaluate("СУММ(B2:B5)")
```

При этом можно использовать внешние ссылки, то есть ссылки на другие книги. Например:

```
Workbooks("BOOK2.XLS").Sheets("МоиДанные").Evaluate("A1").Select
```

Нетрудно заметить, что квадратные скобки эквивалентны вызову метода Evaluate со строковым аргументом. Они позволяют сократить код, но не могут быть использованы совместно со строковыми переменными. Например, можно написать:

```
Dim sFormula As String
sFormula = "СУММ(B2:B5)"
MsgBox Evaluate(sFormula).
```


но код

```
MsgBox [sFormula]
```

приведет к появлению строки СУММ(B2:B5), как было бы без квадратных скобок.

Метод Goto

Метод `Goto` выделяет указанный пользователем диапазон ячеек в любой книге. Он также может использоваться для выделения процедур Visual Basic. Вот его синтаксис:

```
Application.Goto(Reference, Scroll)
```

Необязательный параметр *Reference* указывает пункт назначения. Это может быть объект `Range`, строка, содержащая ссылки в условных обозначениях R1C1, или строка, содержащая имя процедуры Visual Basic. Если данный аргумент пропущен, будет использовано предыдущее значение параметра.

Необязательный параметр *Scroll* должен иметь значение `True`. Это дает возможность прокрутки и приводит к тому, что левая верхняя ячейка диапазона назначения оказывается в верхнем левом углу окна. По умолчанию этот параметр имеет значение `False`. В результате, если пункт назначения был видимым в окне, он не двигается, если же он был невидим, то появляется в нижней части окна.

Например, для выделения диапазона B5:C6 активного листа можно воспользоваться следующим кодом:

```
Application.Goto Reference:=Range("B5:C6")
```

или

```
Application.Goto Reference:="R5C2:R6C3"
```

Метод `Goto` также используется совместно с массивом `PreviousSelections`. В частности, объект `Application` имеет свойство `PreviousSelections`, которое возвращает массив объектов `Range` на основе четырех предыдущих выделенных диапазонов. Вот синтаксис этого свойства:

```
Application.PreviousSelections(Index),
```

где аргумент *Index* является числом от 1 до 4.

При каждом выделении диапазона ячеек с помощью текстового поля `Name`, команды `Go To` (Перейти) меню `Edit` (Правка) или метода `Goto` текущий выделенный диапазон добавляется на верхнюю позицию (номер 1) массива `PreviousSelections`, а все остальные элементы последнего перемещаются на одну позицию вниз. При этом завершающий элемент массива просто удаляется.

Для иллюстрации рассмотрим код:

```
Application.Goto Sheet1.Range("A1")  
ActiveCell.Value = 1  
Application.Goto Sheet2.Range("A1")  
ActiveCell.Value = 2.
```

который заполняет первые ячейки каждого из двух листов, добавляя диапазон ячеек в массив `PreviousSelections` с помощью метода `Goto`.

Следующая строка будет осуществлять выбор между двумя ячейками при повторном выполнении кода:

```
Application.Goto Application.PreviousSelections(1)
```

Методы `Goto` и `Select` отличаются друг от друга по следующим параметрам:

- Оба метода используются для выделения определенного диапазона ячеек, но метод `Select` не производит активизации листа, на котором был выделен диапазон.
- Метод `Select` не имеет аргумента *Scroll*.
- Метод `Select` не добавляет выделенный диапазон в массив `PreviousSelections`.
- Метод `Select` оперирует с аргументом *Replace*.

Метод Quit

Метод `Quit` закрывает приложение Excel, и поэтому он связан с событием `BeforeClose`. Это событие имеет параметр *Cancel*, позволяющий отменить действие. Более подробную информацию об этом событии можно найти в главе 11.

Если метод `Quit` обнаруживает открытые несохраненные книги, отображается диалоговое окно с вопросом, хочет ли пользователь сохранить их. Это можно предотвратить, заранее сохранив все книги или присвоив свойству `DisplayAlerts` значение `False`. Однако в последнем случае несохраненные данные будут утеряны.

Обратите внимание на то, что приложение проверяет свойство `Saved` книги, чтобы определить, требуется ли отображать диалоговое окно с предупреждением. Соответственно, если присвоить этому свойству значение `True`, но не сохранить книгу, перед ее закрытием окно с предупреждением не появится, а несохраненные данные будут утеряны.

Прочие свойства и методы

Перечислим ряд дополнительных свойств и методов объекта `Application`.

Объекты `CellFormat`, `FindFormat` и `ReplaceFormat`

Объект `CellFormat` используется совместно со свойствами `FindFormat` и `ReplaceFormat`. Это позволяет программным способом отыскивать и изменять форматирование ячеек.

Новые свойства `FindFormat` и `ReplaceFormat` объекта `Application` возвращают уникальный объект `CellFormat`. Можно указать свойства форматирования любого из объектов `CellFormat` и затем использовать метод `Replace` объекта `Range` для изменения этого форматирования. При этом форматирование, возвращаемое свойством `FindFormat`, будет заменено форматированием, возвращаемым свойством `ReplaceFormat`.

Например, следующий код изменяет формат ячеек таким образом, что вместо полужирного начертания шрифта используется полужирный курсив. Обратите внимание на то, что код не содержит явного описания объекта `CellFormat`.

```
Sub Example_CellFormat()
```

```
    ' Замена полужирного шрифта полужирным курсивом
```

```
    With Application.FindFormat
```

```
        .Clear
```

```
        .Font.Bold = True
```

```
    End With
```

```
    With Application.ReplaceFormat
```

```
        .Clear
```



```
.Font.Bold = True  
.Font.Italic = True  
End With  
Cells.Replace SearchFormat:=True, ReplaceFormat:=True  
  
End Sub
```

Объект `CellFormat` имеет ряд свойств, связанных с форматированием. Они перечислены ниже. Также в состав этого объекта включен метод `Clear`, снимающий любое форматирование. Все эти свойства используются точно так же, как и свойство `Font` в предыдущем примере кода.

- `AddIndent`
- `Borders`
- `Font`
- `FormulaHidden`
- `HorizontalAlignment`
- `IndentLevel`
- `Interior`
- `Locked`
- `MergeCells`
- `NumberFormat`
- `NumberFormatLocal`
- `Orientation`
- `ShrinkToFit`
- `VerticalAlignment`
- `WrapText`

Метод `InputBox`

Функция `InputBox`, возвращающая введенные пользователем значения, уже рассматривалась ранее. Одноименный метод объекта `Application` также возвращает введенную пользователем информацию, давая при этом возможность проверки типа возвращаемого значения. Кроме того, он возвращает формулы Excel, объекты и значения ошибки.

Вот синтаксис метода `InputBox`:

```
Application.InputBox(Prompt, Title, Default,  
                    Left, Top, HelpFile, HelpContextId, Type)
```

Все параметры данного метода за исключением *Prompt* являются необязательными.

- *Prompt* — сообщение, вводимое в диалоговое окно. Это может быть строка, число, дата или логическое значение.
- *Title* — название диалогового окна. По умолчанию используется значение `Input`.
- *Default* — значение, отображающееся в текстовом поле диалогового окна. Если этот аргумент пропущен, текстовое поле будет пустым.
- *Left* и *Top* — расстояние от верхнего левого угла диалогового окна до верхнего левого угла экрана (в точках).

- *HelpFile* и *HelpContext* — название справочного файла и контекстной тематической ссылки, вызываемой пользователем щелчком на кнопке Help в диалоговом окне, предназначенном для ввода данных. Если эти аргументы пропущены, кнопка Help здесь просто отсутствует.
- *Type* — типы данных, которые могут вводиться пользователем в текстовое поле. Этот параметр также определяет возвращаемый тип данных метода InputBox. Это может быть одно или несколько значений из табл. 16.5. По умолчанию данный аргумент имеет значение 2, что соответствует вводу текста.

Таблица 16.5. Значения параметра Type метода InputBox

Значение	Тип параметра
0	Формула
1	Число
2	Текст (строка)
4	Логическое значение (True или False)
8	Ссылка на ячейку
16	Значение ошибки, например #N/A
64	Массив значений

К сожалению, проверка типа данных в методе InputBox иногда осуществляется не совсем корректно. Для иллюстрации рассмотрим оператор:

```
Range("A1").Value = Application.InputBox( _  
    Prompt:="Enter data", Type:=0)
```

Появляющееся в результате диалоговое окно должно допускать исключительно ввод формул. Однако если ввести в него текст, он просто появится в ячейке A1. В справочной документации сказано, что если параметр *Type* имеет значение 0, метод InputBox возвращает формулу как текст, а все ссылки в этой формуле преобразуются в ссылки вида A1.

Если параметр *Type* имеет значение 8, метод InputBox возвращает объект Range. Следовательно, для назначения объекта переменной типа Range требуется использовать оператор Set:

```
Dim rng as Variant  
Set rng = Application.InputBox(  
    Prompt:="Enter Cell Reference", Type:=8)
```

При пропуске этого оператора переменной типа Range вместо объекта Range будет присвоен некий диапазон. Если указать в описании, что переменная rng принадлежит к типу Range, пропуск в вышеприведенном коде оператора Set приведет к появлению сообщения об ошибке «Object variable or With block variable not set».

Если переменная *Type* имеет значение 64, пользователь должен ввести два ряда данных, которые будут рассматриваться как двумерный массив. В качестве примера рассмотрим лист, показанный на рис. 16.3.

Для кода

```
Dim a As Variant  
a = Application.InputBox(  
    Prompt:="Enter Array", Type:=64)  
Debug.Print a(3,2)
```

допустим ввод данных A1:B6, после которого переменная a(3,2) будет иметь значение 56.

	A	B
1	4	34
2	5	45
3	23	56
4	14	57
5	55	78
6	67	687

Рис. 16.3. Переменная Type имеет значение 64

В качестве заключительного примера рассмотрим код:

```
Dim a As Variant
a = Application.InputBox( Prompt:="Введите формулу". Type:=1)
Range("D1").Formula = a
```

Если ввести в текстовое поле появляющегося диалогового окна формулу, она не появится в ячейке D1 (туда будет помещен только численный результат).

Свойство Selection

Свойство Selection возвращает выделенный в данный момент объект активного листа. Например, при выделении ячейки свойство возвращает соответствующий ей объект Range. При отсутствии выделенных объектов свойство Selection возвращает значение Nothing. Если применить его к объекту Window, результатом будут объекты, выделенные в активном окне.

Свойство StatusBar (чтение/запись, тип String)

Свойство StatusBar возвращает или устанавливает текст, показываемый в строке состояния Excel. Чтобы вернуть управление строкой состояния приложению, достаточно присвоить этому свойству значение False. Верно также обратное утверждение, что если приложение в данный момент контролирует строку состояния, свойство StatusBar возвращает значение False.

Метод Intersect

Метод Intersect возвращает объект Range в виде прямоугольного пересечения двух или более диапазонов. Вот его синтаксис:

```
Application.Intersect(Arg1, Arg2, ...),
```

где Arg1, Arg2, ... являются объектами Range, пересечение диапазонов которых требуется получить. Например, строка кода:

```
Application.Intersect(Range("A1:D5"), Range("B2:F9")).Select
```

выделяет диапазон B2:D5.

Метод Union

В отличие от метода Intersect метод Union возвращает объединение двух или более диапазонов. Вот его синтаксис:

```
Application.Union(Arg1, Arg2, ...),
```


где *Arg1*, *Arg2*, ... являются объектами *Range*, диапазоны которых требуется объединить. В данном случае требуется указать по меньшей мере два таких объекта. Например, следующий код:

```
Application.Union(Range("A1:D5"), Range("B2:F9")).Select
```

выделяет две прямоугольные области A1:D5 и B2:F9.

Потомки объекта Application

На рис. 16.4 показаны потомки объекта Application.

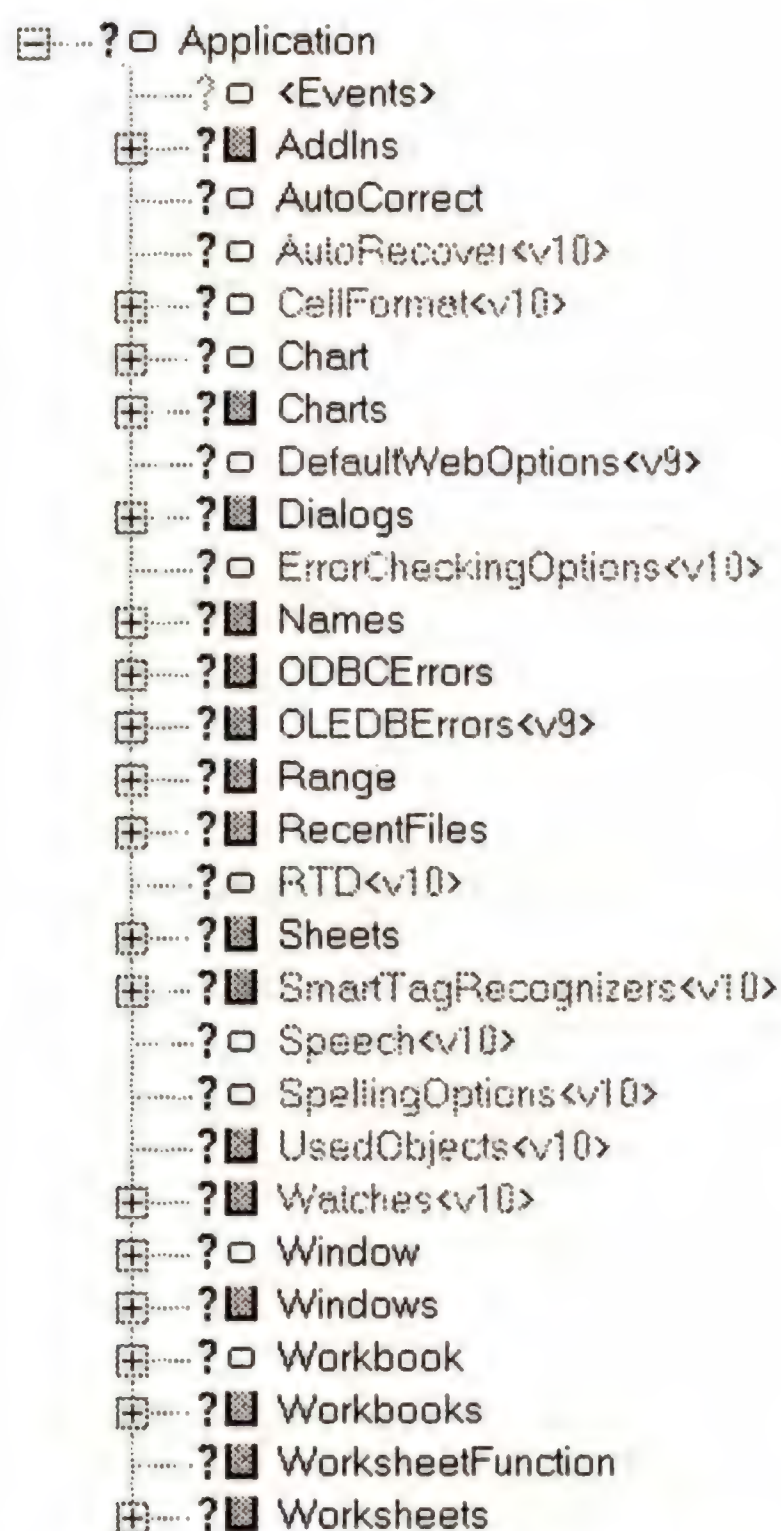


Рис. 16.4. Потомки объекта Application

Большинство этих дочерних объектов, например AppEvents, Chart, Range, Sheets, Workbook и Worksheet, будут рассматриваться в последующих главах. С объектом Dialog вы уже познакомились. В данной главе мы поговорим о менее значимых потомках объекта Application.

Объект Name и семейство Names

Объект Name представляет собой имя диапазона ячеек. В Excel существует два типа имен: системные (например, Print_Area) и пользовательские.

Объекты Name хранятся в нескольких семействах Names. Существует семейство Names как для объекта Application, так и для каждого объекта Workbook и Worksheet.

Создать объект Name можно различными способами. Например, добавить его в семейство Names, вызвав это семейство с помощью метода Add. Или использовать метод CreateNames объекта Range.

К примеру, следующий код:

```
Workbooks("Book1.xls").Names.Add Name:="WkBkName" RefersTo:="=Sheet1!$A$1:$B$1"
```

приводит к появлению объекта Name, связанного с диапазоном на листе Sheet1 книги Book1. Этот объект добавляется в семейство Names книги, а не листа.

Обратите внимание на то, как именно определяется лист в параметре RefersTo и как именно указывается абсолютный адрес. Если оператор \$ не используется, определение диапазона осуществляется по отношению к активной в данный момент ячейке.

Вот пример кода, добавляющего объект Name в семейства Names листов Sheet1 и Sheet2:

```
Workbooks("Book1.xls").Worksheets("Sheet1"). _  
    .Names.Add Name:="WkSheet1Name", _  
    RefersTo:="=Sheet1!$A$1:$B$1"  
Workbooks("Book1.xls").Worksheets("Sheet2"). _  
    Names.Add Name:="WkSheet2Name", _  
    RefersTo:="=Sheet2!$A$1:$B$1"
```

Впрочем, этот код также добавляет объект Name в семейство Names активной книги.

Данная строка кода устанавливает для диапазона WkSheet1Name полужирный шрифт:

```
Sheet1.Names("WkSheet1Name").RefersToRange.Font.Bold = True
```

Помните, что для данного объекта Range не существует семейства Names, хотя объекты этого типа могут иметь и несколько имен. Лучше всего взять для диапазона первое имя, используя для этого свойство Name.

Рассмотрим ряд свойств и методов объекта Name:

- Метод Delete удаляет объект Name из семейства Names, к которому объект принадлежит. При этом удаления самого диапазона не происходит. Вот синтаксис этого метода:
NameObject.Delete
- Свойство Name возвращает или устанавливает имя объекта Name.
- Свойство RefersTo возвращает или задает адрес диапазона или формулы в стиле A1 и начинается со знака равно.
- Свойство RefersToR1C1 является ссылкой на имя в стиле R1C1.
- Свойство RefersToRange возвращает объект Name, связанный с именованным диапазоном.
- Свойство Value эквивалентно свойству RefersTo, так как тоже возвращает или задает адрес диапазона или формулы в стиле A1 и начинается со знака равенства.
- Свойство Visible возвращает или устанавливает видимость именованного диапазона.

Семейство Windows и объекты Window

В соответствии со своим названием объект Window представляет собой окно Excel. Семейство Windows объекта Application включает в себя все открытые книги выполняющегося в данный момент приложения. Соответственно, семейство Windows объекта Workbook содержит только окна открытых книг.

Метод Arrange семейства Windows используется для компоновки окон на экране. Вот его синтаксис:

```
WindowsObject.Arrange(ArrangeStyle, _
    ActiveWorkbook, SyncHorizontal, SyncVertical)
```

Необязательный параметр *ArrangeStyle* может принимать значения из следующего перечисления:

```
Enum XlArrangeStyle
    xlArrangeStyleVertical = -4166
    xlArrangeStyleHorizontal = -4128
    xlArrangeStyleTiled = 1 ' Значение по умолчанию
    xlArrangeStyleCascade = 7
End Enum
```

Присвоение параметру *ActiveWorkbook* значения True приводит к тому, что компонуются только видимые окна активной книги. По умолчанию этот параметр имеет значение False, то есть компоновке подвергаются все открытые окна.

Остальные параметры играют роль только в том случае, когда параметр *ActiveWorkbook* имеет значение True. Присвоение параметру *SyncHorizontal* значения True позволяет синхронизировать горизонтальную прокрутку окон активной книги. По умолчанию этот параметр имеет значение False. Соответственно, параметр *SyncHorizontal* отвечает за прокрутку окон по горизонтали. Вот код, который располагает на экране видимые окна и синхронизирует прокрутку по горизонтали:

```
ActiveWorkbook.Windows.Arrange
    ArrangeStyle:=xlArrangeStyleTiled, _
    SyncHorizontal:=True
```

Для создания нового окна используется метод NewWindow объекта Workbook:

```
ThisWorkbook.NewWindow
```

В результате появляется копия активного окна. Этот метод также применим к любому существующему объекту Window для получения его копии.

Остальные 58 свойств и методов объекта Window перечислены в табл. 16.6.

Таблица 16.6. Свойства и методы объекта Window

_DisplayRightToLeft<v9>	EnableResize	ScrollWorkbookTabs
Activate	FreezePanes	SelectedSheets
ActivateNext	GridlineColor	Selection
ActivatePrevious	GridlineColorIndex	SmallScroll
ActiveCell	Height	Split
ActiveChart	Index	SplitColumn
ActivePane	LargeScroll	SplitHorizontal
ActiveSheet	Left	SplitRow
Application	NewWindow	SplitVertical
Caption	OnWindow	TabRatio
Close	Panes	Top

Creator	Parent	Type
DisplayFormulas	PointsToScreenPixelsX<v9>	UsableHeight
DisplayGridlines	PointsToScreenPixelsY<v9>	UsableWidth
DisplayHeadings	PrintOut	View
DisplayHorizontalScrollBar	PrintPreview	Visible
DisplayOutline	RangeFromPoint<v9>	VisibleRange
DisplayRightToLeft	RangeSelection	Width
DisplayVerticalScrollBar	ScrollColumn	WindowNumber
DisplayWorkbookTabs	ScrollIntoView<v9>	WindowState
DisplayZeros	ScrollRow	Zoom

Объект WorksheetFunction

Объект WorksheetFunction возвращается одноименным свойством объекта Application. Его единственным назначением является обеспечение доступа к функциям листа Excel. Например, следующий код иллюстрирует доступ к функции Min с помощью объекта WorksheetFunction:

```
Dim rng As Range
Dim rMin As Single
Set rng = Worksheets("Sheet1").Range("A1:D10")
rMin = Application.WorksheetFunction.Min(rng)
```


17. Объект Workbook

В этой главе мы поговорим об объекте `Workbook` и семействе `Workbooks`. На рис. 17.1 показана часть объектной модели Excel, связанная непосредственно с книгами (`Workbook`).

Семейство `Workbooks`

Объект `Application` включает свойство `Workbooks`, которое возвращает семейство `Workbooks`, содержащее все объекты `Workbook`, открытые в данный момент в Excel. Вот пример кода, позволяющего узнать количество открытых книг:

```
Dim wbs As Workbooks
Set wbs = Application.Workbooks
MsgBox wbs.Count
```

Рассмотрим ряд свойств и методов семейства `Workbooks`.

Метод `Add`

Метод `Add` приводит к появлению новой книги, которая добавляется в семейство `Workbooks`. Новая книга автоматически становится активной. Вот синтаксис этого метода:

```
WorkbooksObject.Add(Template).
```

где необязательный параметр *Template* определяет способ создания новой книги. Если в качестве этого аргумента фигурирует имя существующего шаблона Excel, новый файл будет создан на основе данного шаблона.

Шаблоны в Excel могут иметь определенное содержимое (например, ярлыки строк и столбцов), форматирование, макросы, а также собственные меню и панели инструментов. Соответственно, любая книга, основанная на шаблоне, наследует все содержимое последнего.

Аргумент *Template* также может принимать значения констант из следующего перечисления:

```
Enum XlWBATemplate
    xlWBATWorksheet = -4167
    xlWBATChart = -4109
    xlWBATExcel4MacroSheet = 3
    xlWBATExcel4IntlMacroSheet = 4
End Enum
```

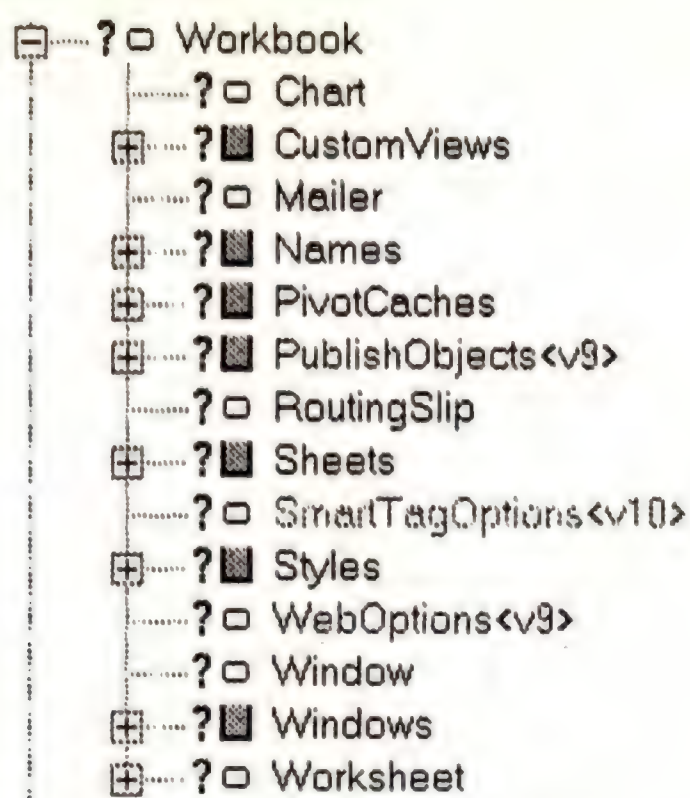



Рис. 17.1. Объект Workbook

В этом случае новая книга будет содержать всего один лист указанного типа. Если пропустить аргумент *Template*, Excel создаст новую книгу, количество листов в которой определяется свойством *SheetsInNewWorkbook* объекта *Application*.

Метод Close

Метод *Close* закрывает все открытые книги. Вот его синтаксис:

```
WorkbooksObject.Close
```

Свойство Count

Большинство семейств имеют свойство *Count*. Семейство *Workbooks* не является исключением. Это свойство просто возвращает число книг, открытых в данный момент.

Свойство Item

Свойство *Item* возвращает определенную книгу семейства *Workbooks*. Например, код

```
Workbooks.Item(1)
```

возвращает объект *Workbook*, соответствующий первой книге семейства *Workbooks*. Так как свойство *Item* присуще данному семейству по умолчанию, можно записать просто:

```
Workbooks(1)
```

Имейте в виду, что нельзя полагаться на тот факт, что какая-либо книга имеет определенный номер. Впрочем, это утверждение применимо ко всем семействам. Соответственно, для доступа к определенной книге нужно использовать не номер, а ее имя:

```
Workbooks("Book1.xls")
```

Важно понимать, что при создании пользователем новой книги, например, с именем *Book2* с помощью команды *New* (Создать) меню *File* (Файл) ссылка на книгу должна выглядеть следующим образом:

```
Workbooks("Book2")
```


А запись:

```
Workbooks("Book2.xls")
```

будет вызывать сообщение об ошибке, пока книга не сохранена на диске.

Метод Open

Метод Open открывает существующие книги. Его синтаксис достаточно сложен:

```
WorkbooksObject.Open(FileName, UpdateLinks, ReadOnly, _  
    Format, Password, WriteResPassword, IgnoreReadOnlyRecommended, _  
    Origin, Delimiter, Editable, Notify, Converter, AddToMRU)
```

Впрочем, большинство этих параметров используется довольно редко. Например, некоторые из них задействованы при открытии текстовых файлов. Информацию о них можно найти в справочной документации Excel. Кроме того, все они, за исключением параметра *FileName*, являются необязательными.

Параметр *FileName* представляет собой имя файла открываемой книги. Чтобы открыть книгу только для чтения, нужно присвоить параметру *ReadOnly* значение *True*.

Для защиты книги паролем используется параметр *Password*. Если пароль для открытия книги не указан, приложение попросит вас указать его.

Чтобы добавить книгу в список последних открываемых файлов, достаточно присвоить параметру *AddToMru* значение *True*. По умолчанию этот параметр имеет значение *False*.

Метод OpenText

Метод OpenText открывает текстовый файл, как новую книгу. Текстовые данные разбираются и помещаются на отдельный лист. Вот синтаксис этого метода:

```
WorkbooksObject.OpenText(Filename, Origin, StartRow, _  
    DataType, TextQualifier, ConsecutiveDelimiter, Tab, _  
    Semicolon, Comma, Space, Other, OtherChar, FieldInfo)
```

Все параметры данного метода, кроме *FileName*, являются необязательными. Параметр *FileName* указывает имя текстового файла, который требуется открыть.

Параметр *Origin* определяет происхождение текстового файла и может принимать значения констант из следующего перечисления:

```
Enum XlPlatform  
    xlMacintosh = 1  
    xlWindows = 2  
    xlMSDOS = 3  
End Enum
```

Значение *xlWindows* соответствует текстовому файлу ANSI, в то время как константа *xlMSDOS* указывает на файл ASCII. Если данный аргумент пропущен, выбор параметра File Origin (Формат исходных данных) производится с помощью Text Import Wizard (Мастера импорта текстов).

Параметр *StartRow* указывает номер строки, с которой начинается разбор текста в текстовом файле. По умолчанию этот параметр имеет значение 1.

Необязательный параметр *DataType* определяет формат текста в файле и может принимать значения констант из следующего перечисления:

```
Enum XlTextParsingType  
    xlDelimited = 1 ' Значение по умолчанию
```



```
xlFixedWidth = 2
End Enum
```

Параметр *TextQualifier* используется для описания текста и принимает значения констант из следующего перечисления:

```
Enum xlTextQualifier
    xlTextQualifierNone = -4142
    xlTextQualifierDoubleQuote = 1 ' Значение по умолчанию
    xlTextQualifierSingleQuote = 2
End Enum
```

Если требуется, чтобы Excel рассматривал последовательные разделители как один, параметру *ConsecutiveDelimiter* требуется присвоить значение True. По умолчанию он имеет значение False.

Существует ряд параметров, для которых требуется, чтобы параметр *DataType* имел значение xlDelimited. Если какой-нибудь из этих параметров имеет значение True, это значит, что Excel должен рассматривать соответствующий символ как разделитель текста. Перечислим эти параметры, имеющие по умолчанию значение False:

- *Tab*. Когда этот параметр принимает значение True, в качестве разделителя используется символ табуляции.
- *Semicolon*. Когда этот параметр принимает значение True, в качестве разделителя используется точка с запятой.
- *Comma*. Когда этот параметр принимает значение True, в качестве разделителя используется запятая.
- *Space*. Когда этот параметр принимает значение True, в качестве разделителя используется пробел.
- *Other*. Когда этот параметр принимает значение True, в качестве разделителя используется знак, определенный аргументом *OtherChar*. Если этот аргумент содержит несколько знаков, разделителем служит первый из них.

Параметр *FieldInfo* является массивом, который содержит информацию, необходимую для разбора отдельных столбцов данных файла-источника. Интерпретация этого параметра зависит от значения параметра *DataType*.

Если последний параметр имеет значение xlDelimited, аргумент *FieldInfo* будет представлять собой массив, размер которого определяется числом столбцов преобразуемых данных. Первый элемент двухэлементного массива представляет собой номер столбца (нумерация начинается с номера 1), а второй — номер, указывающий на способ разбора столбца.

Значение	Описание
1	Общий
2	Текст
3	Данные МДГ
4	Данные ДМГ
5	Данные ГМД
6	Данные МГД
7	Данные ДГМ
8	Данные ГДМ
9	Пропустить столбец

Если для рассматриваемого столбца отсутствует двухэлементный массив, разбор столбца производится общим образом. Например, следующее значение параметра *FieldInfo*:

```
Array(Array(1, 2), Array(3, 9))
```

приводит к тому, что первый столбец разбирается как текст, а третий — пропускается. Все остальные столбцы разбираются как общие данные.

Для иллюстрации рассмотрим файл *temp.txt*, расположенный в папке *excel* на диске *D:*, который содержит следующий текст:

```
"John","Smith","Serial Record",1/2/98
"Fred","Gwynn","Serials Other Dept",2/2/98
"Mary","Davis","English Dept",3/5/98
"David","John","Chemistry Dept",4/4/98
```

Код

```
Workbooks.OpenText _
  FileName:="d:\excel\temp.txt", _
  Origin:=xlMSDOS, _
  StartRow:=1, _
  DataType:=xlDelimited, _
  TextQualifier:=xlTextQualifierDoubleQuote, _
  ConsecutiveDelimiter:=True, _
  Comma:=True, _
  FieldInfo:=Array(Array(1, 2), _
    Array(2, 2), Array(3, 2), Array(4, 6))
```

приводит к появлению листа, показанного на рис. 17.2. Обратите внимание на то, что ячейки в столбце D отформатированы как даты.

	A	B	C	D
1	John	Smith	Serial Record	1/2/98
2	Fred	Gwynn	Serials Order Dept	2/2/98
3	Mary	Davis	English Dept	3/5/98
4	David	Johns	Chemistry Dept	4/4/98

Рис. 17.2. Результат открытия текстового файла в Excel. В качестве разделителя использовалась запятая

С другой стороны, если параметр *DataType* имеет значение *xlFixedWidth*, первый элемент любого двухэлементного массива будет указывать начальное положение символа в столбце (при этом первым символом будет 0), а второй, как и в предыдущем случае, будет определять способ разбора текста.

Для иллюстрации рассмотрим текстовый файл следующего содержания:

```
0-125-689
2-523-489
3-424-664
4-125-160
```

Код

```
Workbooks.OpenText _
  FileName:="d:\excel\temp.txt", _
  Origin:=xlMSDOS, _
```



```
StartRow:=1.  
DataType:=xlFixedWidth.  
FieldInfo:=Array(Array(0, 2),  
    Array(1, 9), Array(2, 2), Array(5, 9),  
    Array(6, 2))
```

приводит к появлению листа, показанного на рис. 17.3. Обратите внимание на то, как были использованы массивы для удаления дефисов.

	A	B	C
1	0	125	689
2	2	523	489
3	3	424	664
4	4	125	160

Рис. 17.3. Текстовый файл фиксированной ширины, открытый в Excel

Важно понимать, что открытый в Excel текстовый файл не преобразуется сам по себе в книгу Excel. Для подобного преобразования можно воспользоваться методом SaveAs. Например,

```
Application.ActiveSheet.SaveAs  
    FileName:="d:\excel\temp.xls",  
    FileFormat:=xlWorkbookNormal
```

Объект Workbook

Объект Workbook представляет любую открытую книгу Excel. Как уже упоминалось, все объекты Workbook имеют отношение к семейству Workbooks.

Этот объект имеет 103 свойства и метода, которые перечислены в табл. 17.1.

Таблица 17.1. Свойства и методы объекта Workbook

_CodeName	FullName	RefreshAll
_PrintOut<v9>	FullNameURLEncoded<v10>	RejectAllChanges
_Protect<v10>	HasMailer	ReloadAs<v9>
_ReadOnlyRecommended<v10>	HasPassword	RemovePersonalInformation<v10>
_SaveAs<v10>	HasRoutingSlip	RemoveUser
AcceptAllChanges	HighlightChangesOnScreen	Reply
AcceptLabelsInFormulas	HighlightChangesOptions	ReplyAll
Activate	HTMLProject<v9>	ReplyWithChanges<v10>
ActiveChart	IsAddin	ResetColors
ActiveSheet	IsInplace	RevisionNumber
AddToFavorites	KeepChangeHistory	Route
Application	Keywords	Routed
Author	LinkInfo	RoutingSlip
AutoUpdateFrequency	LinkSources	RunAutoMacros

продолжение ➤

Таблица 17.1 (продолжение)

AutoUpdateSaveChanges	ListChangesOnNewSheet	Save
BreakLink<v10>	Mailer	SaveAs
BuiltinDocumentProperties	MergeWorkbook	SaveCopyAs
CalculationVersion<v9>	Modules	Saved
CanCheckIn<v10>	MultiUserEditing	SaveLinkValues
ChangeFileAccess	Name	sblt<v9>
ChangeHistoryDuration	Names	SendForReview<v10>
ChangeLink	NewWindow	SendMail
Charts	OnSave	SendMailer
CheckIn<v10>	OnSheetActivate	SetLinkOnData
Close	OnSheetDeactivate	SetPasswordEncryptionOptions<v10>
CodeName	OpenLinks	Sheets
Colors	Parent	ShowConflictHistory
CommandBars	Password<v10>	ShowPivotTableFieldList<v10>
Comments	PasswordEncryptionAlgorithm<v10>	SmartTagOptions<v10>
ConflictResolution	PasswordEncryptionFileProperties<v10>	Styles
Container	PasswordEncryptionKeyLength<v10>	Subject
CreateBackup	PasswordEncryptionProvider<v10>	TemplateRemoveExtData
Creator	Path	Title
CustomDocumentProperties	PersonalViewListSettings	Unprotect
CustomViews	PersonalViewPrintSettings	UnprotectSharing
Date1904	PivotCaches	UpdateFromFile
DeleteNumberFormat	PivotTableWizard	UpdateLink
DialogSheets	Post	UpdateLinks<v10>
DisplayDrawingObjects	PrecisionAsDisplayed	UpdateRemoteReferences
Dummy16<v10>	PrintOut	UserControl
Dummy17<v10>	PrintPreview	UserStatus
EnableAutoRecover<v10>	Protect	VBASigned<v9>
EndReview<v10>	ProtectSharing	VBProject
EnvelopeVisible<v9>	ProtectStructure	WebOptions<v9>
Excel4IntlMacroSheets	ProtectWindows	WebPagePreview<v9>
Excel4MacroSheets	PublishObjects<v9>	Windows
ExclusiveAccess	PurgeChangeHistoryNow	Worksheets
FileFormat	ReadOnly	WritePassword<v10>
FollowHyperlink	ReadOnlyRecommended	WriteReserved
ForwardMailer	RecheckSmartTags<v10>	WriteReservedBy

Некоторые из перечисленных свойств и методов существуют исключительно для того, чтобы предоставлять доступ к потомкам объекта Workbook. Эти дочерние объекты показаны на рис. 17.4 и перечислены в табл. 17.2.

Сделаем несколько замечаний по поводу содержимого табл. 17.2. Во-первых, свойство ActiveSheet может возвращать как объект Chart, так и объект Worksheet. Все зависит от активного на данный момент объекта.

Во-вторых, свойства Charts, Sheets и Worksheets возвращают различные семейства Sheets. В частности, свойство Charts возвращает семейство Sheets, которое содержит все листы с диаграммами, имеющиеся в книге. Сюда не входят внедренные диаграммы. Свойство Worksheets возвращает семейство Sheets, содержащее все листы открытой книги. И наконец, свойство Sheets возвращает семейство Sheets, включающее все листы с данными и диаграммами. Это достаточно редкий пример семейства, которое содержит объекты разных типов. Обратите внимание, что в объектной модели Excel отсутствует объект Sheet.

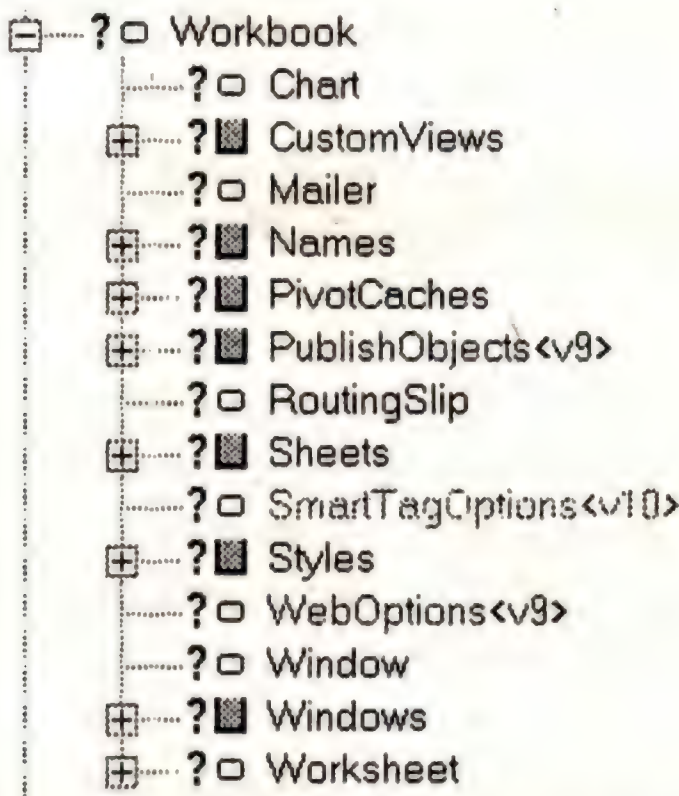


Рис. 17.4. Потомки объекта Workbook

Таблица 17.2. Свойства и методы объекта Workbook, возвращающие дочерние объекты

Name	ReturnType
ActiveChart	Chart
Application	Application
Charts	Sheets
CustomViews	CustomViews
DialogSheets	Sheets
Excel4IntlMacroSheets	Sheets
Excel4MacroSheets	Sheets
Mailer	Mailer
Modules	Sheets
Names	Names
NewWindow	Window
PivotCaches	PivotCaches
PublishObjects	PublishObjects
RoutingSlip	RoutingSlip
Sheets	Sheets
SmartTagOptions	SmartTagOptions
Styles	Styles
WebOptions	WebOptions
Windows	Windows
Worksheets	Sheets

Метод Activate

Метод Activate активизирует книгу. Его синтаксис достаточно прямолинеен:

```
Workbooks("MyWorkBook").Activate
```

Так как параметр Workbook является глобальным, ключевое слово Application можно пропустить.

Метод Close

Метод Close закрывает книги Excel. Вот его синтаксис:

```
WorkbookObject.Close(SaveChanges, FileName, RouteWorkbook)
```

Учтите, что метод Close объекта Workbook имеет три параметра в отличие от метода Close семейства Workbooks, который не имеет параметров вообще.

Необязательный параметр *SaveChanges* используется для сохранения изменений перед закрытием книги. Если никаких изменений в книгу внесено не было, этот параметр игнорируется. Он также игнорируется, если книга открыта в нескольких окнах. С другой стороны, если в закрываемом окне книги присутствуют уникальные изменения, данный параметр производит нужный эффект.

Если параметр *SaveChanges* имеет значение True, изменения сохраняются. Если книга еще не имеет имени (то есть ранее ее не сохраняли), используется имя, заданное параметром *FileName*. При условии, что этот параметр пропущен, пользователю предлагается указать имя. Если параметр *SaveChanges* имеет значение False, сохранения изменений не происходит. Ну и наконец, при пропуске данного аргумента появится диалоговое окно с вопросом, хотите ли вы сохранить изменения.

Необязательный параметр *RouteWorkbook* связан с путем доступа к файлу. Дополнительную информацию о нем можно найти в справочной системе VBA.

Метод Close проверяет состояние свойства Saved книги, чтобы определить, нужно ли предлагать пользователю сохранить изменения. Если присвоить данному свойству значение True, книга будет просто закрыта без предупреждения о несохраненных изменениях.

Свойство DisplayDrawingObjects

Свойство DisplayDrawingObjects возвращает или устанавливает значение, указывающее на способ показа форм. Оно может принимать значения констант из следующего перечисления:

```
Enum xlDisplayShapes  
    xlDisplayShapes = -4104  
    xlPlaceholders = 2  
    xlHide = 3
```

Значение xlDisplayShapes соответствует видимым формам, значение xlPlaceholders приводит к тому, что вместо форм отображаются элементы электронного шаблона, а последнее значение xlHide указывает на тот факт, что формы будут скрыты.

Свойство FileFormat (только чтение, тип Long)

Свойство FileFormat возвращает формат файла или тип книги. Оно принимает значения констант из следующего перечисления:


```
Enum XlFileFormat
    xlAddIn = 18
    xlCSV = 6
    xlCSVMac = 22
    xlCSVMSDOS = 24
    xlCSVWindows = 23
    xlCurrentPlatformText = -4158
    xlDBF2 = 7
    xlDBF3 = 8
    xlDBF4 = 11
    xlDIF = 9
    xlExcel2 = 16
    xlExcel2FarEast = 27
    xlExcel3 = 29
    xlExcel4 = 33
    xlExcel4Workbook = 35
    xlExcel5 = 39
    xlExcel7 = 39
    xlExcel9795 = 43
    xlHtml = 44
    xlIntlAddIn = 26
    xlIntlMacro = 25
    xlSYLK = 2
    xlTemplate = 17
    xlTextMac = 19
    xlTextMSDOS = 21
    xlTextPrinter = 36
    xlTextWindows = 20
    xlUnicodeText = 42
    xlWebArchive = 45
    xlWJ2WD1 = 14
    xlWJ3 = 40
    xlWJ3FJ3 = 41
    xlWK1 = 5
    xlWK1ALL = 31
    xlWK1FMT = 30
    xlWK3 = 15
    xlWK3FM3 = 32
    xlWK4 = 38
    xlWKS = 4
    xlWorkbookNormal = -4143
    xlWorks2FarEast = 28
    xlWQ1 = 34
    xlXMLSpreadsheet = 46
End Enum
```

Свойства Name, FullName и Path

Свойство Name возвращает имя книги, свойство Path возвращает путь доступа к файлу книги, а свойство FullName возвращает полное имя файла (вместе с путем доступа). Все эти свойства предназначены только для чтения.

Если использовать свойство Path без уточнения, а именно в виде:

Application.Path.

будет возвращен маршрут доступа к приложению Excel, а не к определенной книге.

Свойство HasPassword (только чтение, тип Boolean)

Свойство HasPassword, предназначенное только для чтения, имеет значение True, если книга защищена паролем. Пароль может быть назначен в качестве одного из параметров метода SaveAs.

Свойство PrecisionAsDisplayed (чтение/запись, тип Boolean)

Если свойство PrecisionAsDisplayed имеет значение True, вычисления в книге производятся с той точностью, с которой числа показаны на экране, а не с той, с которой они хранятся в памяти. По умолчанию это свойство имеет значение False, то есть для вычислений берутся хранимые в памяти значения.

Метод PrintOut

Метод PrintOut используется для вывода на печать всей книги. Этот метод применим и к ряду других объектов, в частности к объектам Range, Worksheet и Chart. Вот его синтаксис:

```
WorkbookObject.PrintOut(From, To, Copies,  
    Preview, ActivePrinter, PrintToFile, Collate)
```

Все параметры данного метода являются необязательными.

Параметр *From* указывает номер первой распечатываемой страницы, а параметр *To*, соответственно, — номер последней страницы. Пропуск этих параметров приводит к выводу на печать книги (или другого объекта) целиком.

Параметр *Copies* определяет количество распечатываемых копий. По умолчанию он имеет значение 1.

Параметру *Preview* желательно присвоить значение True, чтобы получить возможность предварительного просмотра объекта перед его выводом на печать. По умолчанию он имеет значение False.

Параметр *ActivePrinter* определяет имя используемого принтера. При этом присвоение параметру *PrintToFile* значения True приводит к тому, что приложение выводит на печать файл. Пользователю будет предложено ввести имя выходного файла. К сожалению, это невозможно сделать непосредственно в коде.

Параметр *Collate* должен иметь значение True, чтобы осуществить сортировку множества многостраничных копий.

Метод PrintPreview

Метод PrintPreview активизирует функцию предварительного просмотра. Вот его синтаксис:

```
WorkbookObject.PrintPreview
```

Метод PrintPreview применяется к тому же набору объектов, что и метод PrintOut.

Метод Protect

Метод Protect защищает книгу от редактирования. Вот его синтаксис:

```
WorkbookObject.Protect(Password, Structure, Windows)
```


Этот метод также применим к диаграммам и листам, но со слегка измененным синтаксисом.

Необязательный параметр *Password* определяет пароль. При этом имеет значение регистр букв. Если данный аргумент пропущен, для возвращения возможности редактирования книги не требуется ввод пароля.

Присвоение необязательному параметру *Structure* значения *True* защищает структуру книги, то есть относительную позицию содержащихся в ней листов. По умолчанию этот параметр имеет значение *False*.

Присвоение необязательному параметру *Window* значения *True* защищает окно книги. По умолчанию этот параметр имеет значение *False*.

Свойство **ReadOnly** (только чтение, тип **Boolean**)

Если свойство *ReadOnly* имеет значение *True*, значит, книга открыта только для просмотра, то есть пользователь не может вносить в нее изменения.

Метод **RefreshAll**

Метод *RefreshAll* восстанавливает все диапазоны внешних данных и сводные таблицы в книге. Вот его синтаксис:

```
WorkbookObject.RefreshAll
```

Метод **Save**

Метод *Save* просто сохраняет все внесенные в книгу изменения. Вот его синтаксис:

```
WorkbookObject.Save
```

Метод **SaveAs**

Метод *SaveAs* сохраняет книгу с внесенными в нее изменениями в определенный файл. Вот его синтаксис:

```
expression.SaveAs(Filename, FileFormat, Password, WriteResPassword, _  
ReadOnlyRecommended, CreateBackup, AccessMode, ConflictResolution, _  
AddToMru, TextCodePage, TextVisualLayout)
```

Параметр *FileName* представляет собой имя сохраняемого на диске файла. Если путь доступа не указан в явном виде, используется текущая папка.

Параметр *FileFormat* указывает формат сохраняемого файла. Он может принимать значения констант из перечисления *XlFileFormat*, описанного при рассмотрении свойства *FileFormat*.

Параметр *Password* задает пароль, используемый при сохранении файла. Это строка, максимальная длина которой составляет 15 символов. Важно помнить, что в данном случае имеет значение регистр букв.

Параметр *WriteResPassword* является строкой, задающей пароль, который защищает файл от записи. Открыв такой файл без ввода пароля, пользователь не сможет редактировать его содержимое.

Если присвоить параметру *ReadOnlyRecommended* значение *True*, при открытии файла появится сообщение с рекомендацией открыть файл только для чтения.

Присвоение параметру *CreateBackup* значения *True* приведет к появлению резервных копий файла.

Параметры *AccessMode* и *ConflictResolution* связаны с совместным доступом к файлу. Дополнительную информацию о них можно найти в справочной документации VBA.

Если параметр *AddToMru* имеет значение *True*, книга добавляется в список последних открывавшихся файлов. По умолчанию этот параметр имеет значение *False*.

Метод **SaveCopyAs**

Метод *SaveCopyAs* сохраняет копию книги в отдельном файле, не затрагивая исходную книгу. Вот его синтаксис:

WorkbookObject.SaveCopyAs(Filename),

где параметр *FileName* задает имя файла, в который копируется исходной книги.

Свойство **Saved** (чтение/запись, тип **Boolean**)

Свойство *Saved* имеет значение *True*, если с момента последнего сохранения книги в нее не было внесено никаких изменений. Обратите внимание на то, что это свойство допускает не только чтение, но и запись, то есть пользователь может присвоить ему значение *True*, несмотря даже на наличие изменений.

Потомки объекта **Workbook**

Итак, обсудим ряд потомков объекта *Workbook*.

Объект **CustomView**

Свойство *CustomView* возвращает семейство *CustomViews*. Каждый объект этого семейства представляет собой пользовательский вариант книги. Так как сущность данных объектов достаточно прозрачна, ограничимся рассмотрением примера. Взгляните на образец листа, показанный на рис. 17.5.

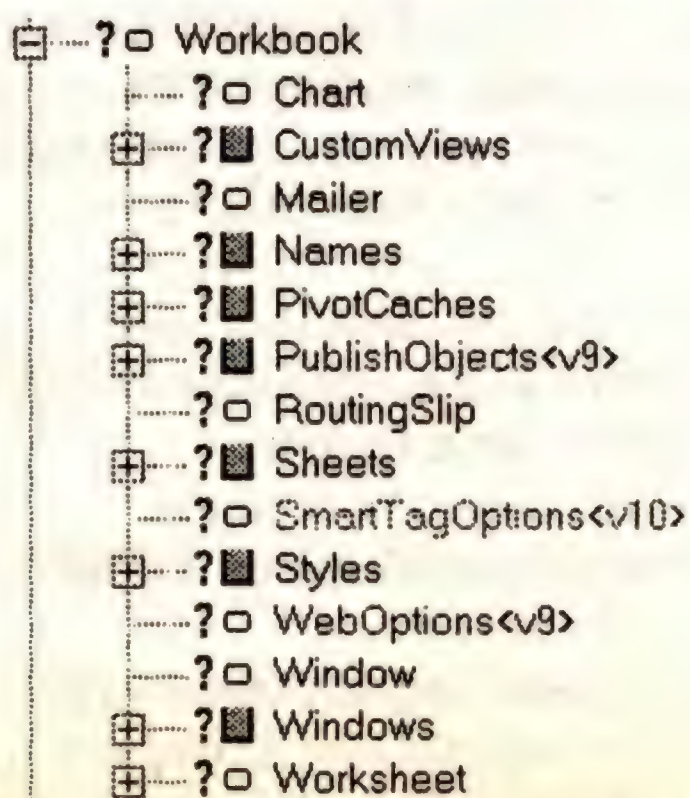


Рис. 17.5. Пример объекта *CustomView*

Теперь выберем команду **Filter ▸ Autofilter** (Фильтр ▸ Автофильтр) в меню **Data** (Данные). Вид листа в результате этой операции показан на рис. 17.6.

	A	B	C
1	Year	ItemCode	Quantity
2	1997	20	30
3	1997	50	60
4	1997	13	90
5	1998	15	56
6	1998	36	67
7	1998	44	78

Рис. 17.6. Вид после фильтрации

Вот код, который присваивает данному пользовательскому варианту книги имя 1998год:

```
ThisWorkbook.CustomViews.Add "1998год"
```

В результате пользователь получает возможность в любой момент перейти к этому варианту книги с помощью кода:

```
ThisWorkbook.CustomViews!1998год.Show
```

или

```
strView = "1998год"  
ActiveWorkbook.CustomViews(strView).Show
```

Семейство Names

Как и объект **Application**, объект **Workbook** имеет свойство **Names**, возвращающее семейство **Names**. Это семейство содержит объекты **Name**, связанные с книгой. Более подробную информацию можно найти в главе 16.

Семейство Sheets

Свойство **Sheets** возвращает семейство **Sheets**, которое содержит объекты **Worksheet**, соответствующие листам с данными, и объекты **Chart**, соответствующие листам с диаграммами. Эти объекты будут подробно обсуждаться в следующих главах.

Семейство Styles и объект Style

Объект **Style** представляет собой набор параметров форматирования для некоего диапазона. Каждая книга имеет семейство **Styles**, содержащее соответствующие ей объекты **Style**.

Для форматирования диапазона достаточно написать:

```
RangeObject.Style = StyleName.
```

где **StyleName** является именем стиля форматирования.

Для создания объекта **Style** применяется метод **Add**, синтаксис которого выглядит следующим образом:

```
WorkbookObject.Add(Name, BasedOn)
```


Обратите внимание, что метод `Add` возвращает вновь созданный объект `Style`.

Параметр *Name* указывает имя стиля, а необязательный параметр *BasedOn* — объект `Range`, форматирование которого используется в качестве основы для нового стиля. Если этот аргумент пропущен, вновь создаваемый стиль будет основан на стиле `Normal`.

Согласно справочной документации, если стиль с указанным именем уже существует, метод `Add` переопределит его на основе параметра *BasedOn*. Однако в моей системе вместо этого появится сообщение об ошибке.

Свойства объекта `Style` связаны с различными функциями форматирования, например названием шрифта, его размером, числовым форматом, выравниванием и т. п. Существует также ряд встроенных стилей, например `Normal` (Обычный), `Currency` (Деньги) или `Percent` (Процент). Названия встроенных стилей можно найти в раскрывающемся списке `Style name` (Имя стиля) диалогового окна `Style` (Стиль), вызываемого командой `Style` (Стиль) меню `Format` (Формат).

Для иллюстрации рассмотрим код, который создает стиль и затем применяет его к произвольному диапазону текущего листа:

```
Dim st As Style
' Удаление существующего стиля
For Each st In ActiveWorkbook.Styles
    If st.Name = "Bordered" Then st.Delete
Next
' Создание нового стиля
With ActiveWorkbook.Styles.Add(Name:="Bordered")
    .Borders(xlTop).LineStyle = xlDouble
    .Borders(xlBottom).LineStyle = xlDouble
    .Borders(xlLeft).LineStyle = xlDouble
    .Borders(xlRight).LineStyle = xlDouble
    .Font.Bold = True
    .Font.Name = "arial"
    .Font.Size = 36
End With
' Применение стиля
Application.ActiveSheet.Range("A1:B3").Style = "Bordered"
```

Пример: сортировка листов книги

Добавим новую служебную программу в приложение `SRXUtils`. При работе с книгами, содержащими множество листов как с данными, так и с диаграммами, иногда требуется отсортировать эти листы в алфавитном порядке.

Основой кода для сортировки листов послужит метод `Move` объектов `Worksheet` и `Chart`. Вот его синтаксис:

```
SheetsObject.Move(Before, After)
```

Разумеется, для эффективного использования данного метода потребуется отсортированный список с именами листов.

Для начала добавим на лист `DataSheet` книги `SRXUtils` новую строку, как показано на рис. 17.7. Расположение строк в данном случае определяется порядком, в котором мы хотим видеть пункты пользовательского меню.

Теперь вставим новый модуль кода с именем `basSortSheets`. Именно он будет содержать код новых процедур. Первая используется для проверки намерения

пользователя произвести сортировку листов. Если пользователь отвечает утвердительно, происходит вызов второй процедуры, которая, собственно, и осуществляет процесс. Первая процедура приведена в листинге 17.1. Она вызывает диалоговое окно, показанное на рис. 17.8.

	A	B	C
1	Year▼	ItemCo▼	Quanti▼
5	1998	15	56
6	1998	36	67
7	1998	44	78

Рис. 17.7. Дополнение к листу DataSheet

Листинг 17.1. Процедура SortSheets

```
Sub SortSheets()  
  If MsgBox("Вы действительно хотите произвести сортировку листов?", _  
    vbOKCancel + vbQuestion, "Сортировка листов") = vbOK Then  
    SortAllSheets  
  End If  
End Sub
```

	A	B	C	D	E	F	G	H
1	Utility	OnAction Proc	Procedure	In Workbook	Menu Item	SubMenu Item	On Wks Menu	On Chart Menu
2	Activate Sheet	RunUtility	ActivateSheet	ThisWorkbook	&Activate Sheet		TRUE	TRUE
3	Print Charts	RunUtility	PrintCharts	Print.utl	&Print	Embedded &Charts	TRUE	TRUE
4	Print Pivot Tables	RunUtility	PrintPivotTables	Print.utl		&Pivot Tables	TRUE	TRUE
5	Print Sheets	RunUtility	PrintSheets	Print.utl		&Sheets	TRUE	TRUE
6	Sort Sheets	RunUtility	SortSheets	ThisWorkbook	&Sort Sheets		TRUE	TRUE

Рис. 17.8. Диалоговое окно, появляющееся в результате выполнения процедуры SortSheets

Процедура, код которой показан в листинге 17.2, собирает имена листов в массив, который помещается на новый лист. Затем с помощью метода Sort, примененного к объекту Range, производится сортировка имен. Происходит повторное заполнение массива, после чего порядок листов изменяется с помощью метода Move.

Листинг 17.2. Процедура SortAllSheets

```
Sub SortAllSheets()  
  
  ' Сортировка листов  
  Dim wb As Workbook  
  Dim ws As Worksheet  
  Dim rng As Range  
  Dim cSheets As Integer  
  Dim sSheets() As String  
  Dim i As Integer  
  
  Set wb = ActiveWorkbook  
  
  ' Определение размерности массива  
  cSheets = wb.Sheets.Count  
  ReDim sSheets(1 To cSheets)  
  
  ' Заполнение массива именами листов  
  For i = 1 To cSheets
```

продолжение ➤

Листинг 17.2 (продолжение)

```
sSheets(i) = wb.Sheets(i).Name
Next

' Создание нового листа и помещение содержимого массива в первый столбец
Set ws = wb.Worksheets.Add
For i = 1 To cSheets
    ws.Cells(i, 1).Value = sSheets(i)
Next

' Сортировка столбца с именами
ws.Columns(1).Sort Key1:=ws.Columns(1), _
    Order1:=xlAscending

' Повторное заполнение массива
For i = 1 To cSheets
    sSheets(i) = ws.Cells(i, 1).Value
Next

' Удаление дополнительного листа
Application.DisplayAlerts = False
ws.Delete
Application.DisplayAlerts = True

' Изменение порядка листов путем перемещения их в конец массива
For i = 1 To cSheets
    wb.Sheets(sSheets(i)).Move After:=wb.Sheets(cSheets)
Next

End Sub
```

После ввода кода достаточно сохранить книгу SRXUtils.xls как надстройку, предварительно отключив уже имеющуюся надстройку с таким именем.

18 **Объект Worksheet**

Объект Worksheet представляет собой лист книги Excel. Соответствующая часть объектной модели данного приложения показана на рис. 18.1.

Свойства и методы объекта Worksheet

Свойства и методы объекта Worksheet перечислены в табл. 18.1.

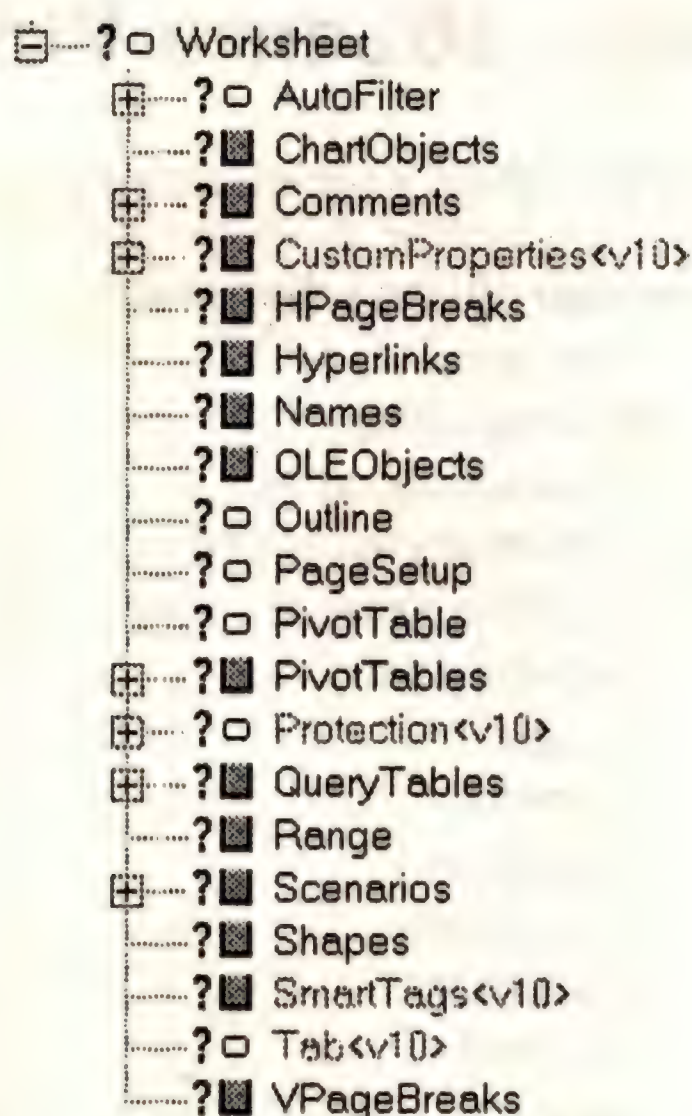
Таблица 18.1. Свойства и методы объекта Worksheet

_CheckSpelling<v9>	DropDowns	Previous
_CodeName	EnableAutoFilter	PrintOut
_DisplayRightToLeft<v9>	EnableCalculation	PrintPreview
_Evaluate	EnableOutlining	Protect
_PasteSpecial<v10>	EnablePivotTable	ProtectContents
_PrintOut<v9>	EnableSelection	ProtectDrawingObjects
_Protect<v10>	Evaluate	Protection<v10>
_SaveAs<v10>	FilterMode	ProtectionMode
Activate	GroupBoxes	ProtectScenarios
Application	GroupObjects	QueryTables
Arcs	HPageBreaks	Range
AutoFilter	Hyperlinks	Rectangles
AutoFilterMode	Index	ResetAllPageBreaks
Buttons	Labels	Rows
Calculate	Lines	SaveAs
Cells	ListBoxes	Scenarios
ChartObjects	MailEnvelope<v10>	Scripts<v9>
CheckBoxes	Move	ScrollArea
CheckSpelling	Name	ScrollBars
CircleInvalid	Names	Select
CircularReference	Next	SetBackgroundPicture
ClearArrows	OLEObjects	Shapes
ClearCircles	OnCalculate	ShowAllData
CodeName	OnData	ShowDataForm

продолжение ➤

Таблица 18.1 (продолжение)

Columns	OnDoubleClick	SmartTags<v10>
Comments	OnEntry	Spinners
ConsolidationFunction	OnSheetActivate	StandardHeight
ConsolidationOptions	OnSheetDeactivate	StandardWidth
ConsolidationSources	OptionButtons	Tab<v10>
Copy	Outline	TextBoxes
Creator	Ovals	TransitionExpEval
CustomProperties<v10>	PageSetup	TransitionFormEntry
Delete	Parent	Type
DisplayAutomaticPageBreaks	Paste	Unprotect
DisplayPageBreaks	PasteSpecial	UsedRange
DisplayRightToLeft	Pictures	Visible
DrawingObjects	PivotTables	VPageBreaks
Drawings	PivotTableWizard	

**Рис. 18.1.** Объект Worksheet

Многие свойства и методы, перечисленные в табл. 18.1, предназначены исключительно для предоставления доступа к потомкам объекта Worksheet. Они отдельно перечислены в табл. 18.2.

Таблица 18.2. Свойства и методы, возвращающие потомков объекта Worksheet

Name	ReturnType
Application	Application
AutoFilter	AutoFilter
Cells	Range
CircularReference	Range

Columns	Range
Comments	Comments
CustomProperties	CustomProperties
HPageBreaks	HpageBreaks
Hyperlinks	Hyperlinks
Names	Names
Outline	Outline
PageSetup	PageSetup
PivotTableWizard	PivotTable
Protection	Protection
QueryTables	QueryTables
Range	Range
Rows	Range
Shapes	Shapes
SmartTags	SmartTags
Tab	Tab
UsedRange	Range
VpageBreaks	VpageBreaks

- Обсудим некоторые свойства и методы из табл. 18.1 более подробно.
- Метод **Activate** приводит к активизации листа. Вот его синтаксис:
`ThisWorkbook.Worksheets("Sheet1").Activate`
 - Свойство **AutoFilterMode** имеет значение **True**, если в настоящий момент на листе показаны раскрывающиеся списки **AutoFilter**. Пользователь может присвоить этому свойству значение **False**, чтобы удалить списки, но лишен возможности присвоить значение **True**. Для создания на листе раскрывающихся списков используется метод **AutoFilter**, который будет подробно рассмотрен в главе 19.
 - Метод **Calculate** пересчитывает все формулы рабочего листа. Этот метод также применим к книгам и отдельным диапазонам. Он имеет очень простой синтаксис:
`WorksheetObject.Calculate`
 - Свойство **CodeName** служит для непосредственного обращения к листу без указания его положения в семействе. Его также можно применять к книгам и диаграммам. Кодовые имена могут быть использованы вместо любых выражений, возвращающих лист. Они задаются в окне **Properties**.
Предположим, что имеется лист с кодовым именем **SheetCodeName** и с именем **SheetName**. В результате следующие строки будут иметь один и тот же эффект:
`Worksheets("SheetName").Activate`
`SheetCodeName.Activate`
При создании листа его имя и кодовое имя совпадают. Впоследствии их можно изменять независимо друг от друга. Однако редактирование кодового имени допустимо только в режиме конструирования. В процессе выполнения кода сделать это невозможно.
 - Синтаксис метода **Copy** зависит от объекта, к которому он применяется. Для копирования листа достаточно написать:
`WorksheetObject.Copy(Before, After),`

где параметр *Before* указывает номер листа, перед которым будет помещена копия, а параметр *After* — номер листа, после которого помещается копия. При выполнении операции используется только один из этих параметров. Если ни один из них не указан, копия листа будет помещена в новую книгу.

Для иллюстрации рассмотрим код, копирующий активный лист и помещающий копию в конец списка имеющихся листов:

```
ActiveSheet.Copy After:=Worksheets(Worksheets.Count)
```

- Метод `Delete` удаляет листы книги. Вот его синтаксис:

```
WorksheetObject.Delete
```

- Если свойство `EnableCalculation` (чтение/запись, тип `Boolean`) имеет значение `True`, приложение автоматически пересчитывает формулы на листе, если это требуется. В противном случае пользователь должен активизировать процесс пересчета. Имейте в виду, что сам процесс присвоения свойству значения `True` сопровождается пересчетом.
- Метод `Evaluate` преобразует имена Excel в объекты или значения. Подробно этот метод рассматривался в главе 16.
- Свойство `FilterMode` (только чтение, тип `Boolean`) имеет значение `True`, если лист находится в режиме фильтрации. Соответственно, если, например, на листе имеются раскрывающиеся списки с параметрами фильтрации, но сам по себе процесс не выполняется, свойство `AutoFilterMode` имеет значение `True`, в то время как свойство `FilterMode` имеет значение `False`. Но при первой же фильтрации последнее свойство принимает значение `True`. При этом свойство `FilterMode` также указывает на наличие скрытых рядов, появившихся после фильтрации.
- Метод `Move` позволяет изменить положение листа в книге. Вот его синтаксис:

```
WorksheetObject.Move(Before, After)
```

где параметры *Before* и *After* имеют тот же самый смысл, что и в методе `Copy`.

- Свойство `Name` (чтение/запись, тип `String`) возвращает или задает имя листа.
- Свойство `Names` возвращает семейство `Names`, в которое входят имена всех листов книги. Детальную информацию об объекте `Name` можно найти в главе 16.
- Метод `PasteSpecial` вставляет содержимое буфера обмена на лист, используя специальное форматирование. Вот наиболее распространенный синтаксис этого метода:

```
WorksheetObject.PasteSpecial(Format)
```

где строковый параметр *Format* указывает формат включаемых данных. Например, следующий код реализует вставку данных в формате документа Word:

```
ActiveSheet.PasteSpecial "Microsoft Word Document"
```

Чтобы познакомиться с остальными вариантами параметра *Format*, можно скопировать нужный объект и воспользоваться диалоговым окном `PasteSpecial` (Специальная вставка).

Перед применением метода `PasteSpecial` требуется указать диапазон, в который будет включен новый объект.

- Метод `PrintOut` используется для вывода на печать листа. Этот метод применим к объектам `Workbook` и `Range`. Вот его синтаксис:

```
WorksheetObject.PrintOut(From, To, Copies,  
    Preview, ActivePrinter, PrintToFile, Collate)
```

Все параметры этого метода являются необязательными.

Параметр *From* указывает номер первого выводимого на печать листа, а параметр *To*, соответственно, — номер последнего листа. Их пропуск приводит к выводу на печать всего объекта (диапазона, листа и т. п.)

Параметр *Copies* определяет количество распечатываемых копий. По умолчанию он имеет значение 1.

Присваивание параметру *SetPreview* значения `True` дает возможность осуществлять предварительный просмотр распечатываемого материала. По умолчанию этот параметр имеет значение `False`.

Параметр *ActivePrinter* определяет имя используемого принтера.

Результатом присваивания параметру *PrintToFile* значения `True` является то, что Excel выводит на печать файл. При этом пользователь должен указать имя выходного файла. К сожалению, не существует способа задать это имя внутри кода.

Параметр *Collate* должен иметь значение `True`, чтобы осуществить сортировку множества многостраничных копий.

- Метод `PrintPreview` активизирует функцию предварительного просмотра листа. Вот его синтаксис:

```
WorksheetObject.PrintPreview
```

- Метод `Protect` защищает лист от редактирования. Вот его синтаксис:

```
WorksheetObject.Protect(Password, DrawingObjects, _  
    Contents, Scenarios, UserInterfaceOnly)
```

Обратите внимание на то, что синтаксис для объектов `Worksheet` и `Workbook` слегка отличается.

Необязательный параметр *Password* является строкой, определяющей пароль для листа. Регистр букв в данном случае имеет значение.

Необязательному параметру *DrawingObject* желательно присвоить значение `True`, чтобы обеспечить защиту форм. По умолчанию он имеет значение `False`.

Если требуется защитить от редактирования ячейки листа, параметр *Content* должен иметь заданное по умолчанию значение `True`.

Для защиты сценариев необходимо сохранить заданное по умолчанию значение `True` параметра *Scenarios*.

Метод `Protect` можно использовать для защиты от пользовательского или программного редактирования ячеек листа. В частности, если параметр `UserInterfaceOnly` имеет значение `True`, пользователь сможет вносить изменения в лист только программным образом. С другой стороны, если этот параметр имеет заданное по умолчанию значение `False`, изменить содержимое листа невозможно ни обычным способом, ни программно. Обратите внимание на то, что в данном случае не макросы имеют защиту от редактирования, а лист защищен от действия макросов.

Если параметру `UserInterfaceOnly` присваивается значение `True` в процессе защиты листа, а затем книга сохраняется, при повторном открытии книги будет за-

щищен весь лист, а не только интерфейс. Чтобы снять защиту с листа, оставив защищенным пользовательский интерфейс, нужно повторно применить метод `Protect`, присвоив параметру `UserInterfaceOnly` значение `True`.

- Свойство `ProtectionMode` (только чтение) по умолчанию имеет значение `False`. Если присвоить ему значение `True`, защищенным окажется только пользовательский интерфейс.
- Метод `SaveAs` сохраняет отредактированный лист в отдельном файле. Вот его синтаксис:

```
WorksheetObject.SaveAs(Filename, FileFormat, Password, _
    WriteResPassword, ReadOnlyRecommended, CreateBackup, _
    AddToMru, TextCodePage, TextVisualLayout)
```

Параметр *FileName* указывает имя файла, в котором будет сохранен лист. Если путь доступа не указан, для сохранения будет использована текущая папка.

Параметр *FileFormat* определяет, в каком формате будет сохранен файл. Он может принимать значения констант из перечисления `XlFileFormat`, описанного в главе 17.

Параметр *Password* является строкой, максимальное количество символов которой достигает 15. Регистр букв имеет значение. Это пароль, используемый при сохранении файла.

Параметр *WriteResPassword* задает пароль, защищающий файл от записи. Пользователь может открыть файл без ввода пароля, но не сможет вносить в него изменения.

Если присвоить параметру *ReadOnlyRecommended* значение `True`, при открытии файла появится сообщение с рекомендацией открыть файл только для чтения.

Присвоение параметру *CreateBackup* значения `True` приведет к появлению резервных копий файла.

Если параметр *AddToMru* имеет значение `True`, книга добавляется в список последних открывавшихся файлов. По умолчанию этот параметр имеет значение `False`.

- Свойство `ScrollArea` возвращает или устанавливает диапазон, в котором допустима прокрутка и выделение ячеек. Его значение должно выглядеть, как ссылка в стиле A1. Например, код:

```
ActiveSheet.ScrollArea = "A1:B200"
```

позволяет осуществлять прокрутку и выделение ячеек только в диапазоне A1:B200. Чтобы снять ограничения, в данном случае достаточно присвоить свойству `ScrollArea` пустую строку:

```
ActiveSheet.ScrollArea = ""
```

Важно понимать, что настройка области прокрутки не имеет ничего общего с блокировкой областей окна.

- Метод `Select` используется для выделения листов. Это процесс, отличающийся от активизации листов с помощью метода `Activate`. За один раз можно выделить несколько листов, например, для удаления. Вот синтаксис метода:

```
WorksheetObject.Select(Replace)
```

Присвоение параметру *Replace* значения `True` приводит к замене выделенного в данный момент листа новым, а не к включению нового листа в выделенный набор.

- Метод `SetBackgroundPicture` позволяет использовать различные изображения в качестве фона листов. Вот его синтаксис:

`WorksheetObject.SetBackgroundPicture(FileName)`,

где параметр *FileName* представляет собой имя файла с графическим изображением.

- Метод `ShowDataForm` отображает формы данных, связанные с листом. Он работает без ошибок только в том случае, когда приложение в состоянии определить, что выделенный в данный момент объект является частью списка. Информацию по работе с формами данных можно найти в справочной информации Excel.

Синтаксис данного метода несложен:

`WorksheetObject.ShowDataForm`

Процедура, в которой происходит вызов метода `ShowDataForm`, обычно замедляет свою работу, так как на показ формы данных требуется время. После закрытия формы данных выполнение процедуры возобновится со строки, следующей за вызовом метода `ShowDataForm`. Другими словами, формы данных являются модальными.

На рис. 18.2 показан пример формы данных для листа `WorksheetObject`.

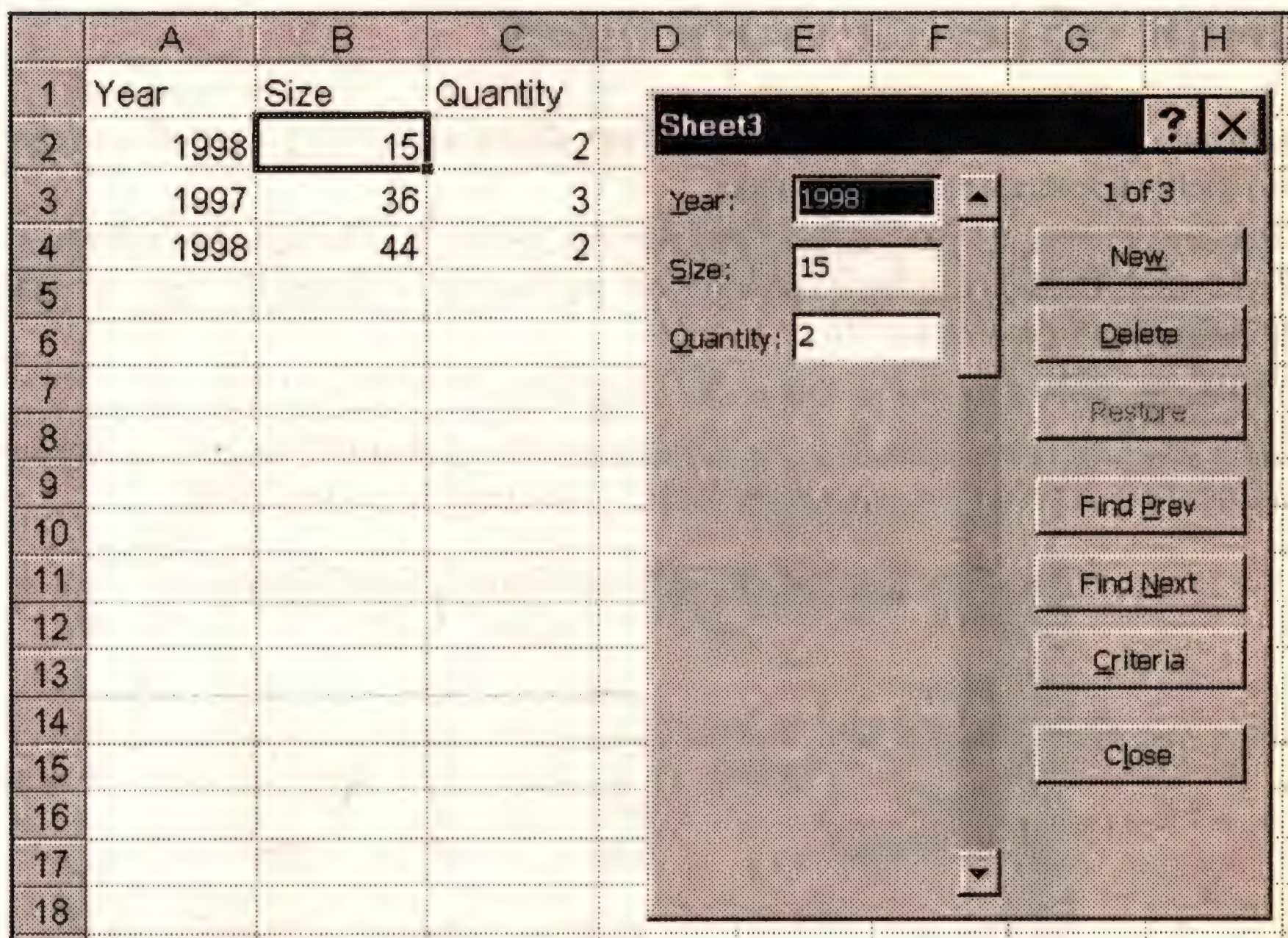


Рис. 18.2. Форма данных

- Метод `Unprotect` снимает защиту с листа. Если лист не защищен паролем, метод не оказывает никакого эффекта. Вот его синтаксис:

`WorksheetObject.Unprotect>Password)`,

где параметр *Password* представляет собой пароль, защищающий лист. При пропуске этого аргумента приложение предлагает пользователю ввести пароль.

- Свойство `UsedRange` возвращает объект `Range`, который представляет наименьшую прямоугольную область, включающую ячейки, используемые в данный момент.

К сожалению, свойство `UsedRange` работало не очень корректно в предыдущих версиях Excel, и авторский опыт подсказывает, что даже в Excel 97 проблема не была решена окончательно. Не существует быстрого способа проверить его надежность, поэтому рекомендуется использовать это свойство крайне аккуратно. Иногда создается впечатление, что в рассмотрение включаются ячейки, которые ранее были заполнены, но потом их содержимое было удалено.

В конце главы 19 будет дан пример функции, позволяющей корректно вычислить используемый диапазон.

- Свойство `Visible` возвращает значение `True`, если лист отображается на экране, и значение `False` в противном случае. Кроме того, оно может принимать значения констант из перечисления `xlVeryHidden`. В последнем случае единственным способом сделать лист видимым является присвоение свойству `Visible` значения `True` в тексте программы.

Потомки объекта Worksheet

Рассмотрим ряд потомков объекта `Worksheet`. Остальные дочерние объекты будут рассмотрены в последующих главах.

- Свойство `Comments` возвращает семейство `Comments`, в которое входят все комментарии (объекты `Comment`), расположенные на листе. Подробно объект `Comment` рассматривается в главе 19.
- Семейство `Names` и объект `Name` уже обсуждались в главе 16.
- Для иллюстрации схематичного представления рассмотрим лист, показанный на рис. 18.3. Требуется получить схему, показанную на рис. 18.4.

	A	B	C	D
1		Col1	Col2	Col3
2	Row1	78	33	23
3	Row2	123	22	222
4	Row3	231	34	345
5	SubTotal1	432	89	590
6				
7	Row4	223	23	454
8	Row5	345	10	53
9	Row6	11	13	4
10	SubTotal2	579	46	511
11				
12	Total	1011	135	1101

Рис. 18.3. Иллюстрация к использованию схем в Excel

1	2	3		A	B	C	D
			1		Col1	Col2	Col3
			2	Row1	78	33	23
			3	Row2	123	22	222
			4	Row3	231	34	345
			5	SubTotal1	432	89	590
			6				
			7	Row4	223	23	454
			8	Row5	345	10	53
			9	Row6	11	13	4
			10	SubTotal2	579	46	511
			11				
			12	Total	1011	135	1101

Рис. 18.4. Конечный результат

Для начала нужно указать свойства объекта Outline для данного листа. Свойство Outline объекта Worksheet возвращает объект Outline, поэтому мы начнем с кода:

```
With ActiveSheet.Outline
    .SummaryRow = xlBelow
    .AutomaticStyles = False
End With
```

Присвоение свойству SummaryRow значения xlBelow говорит о том, что ряды с суммарными данными находятся под обычными данными. Именно там будут помещены кнопки, позволяющие раскрыть/свернуть столбцы с данными.

Присвоение параметру AutomaticStyles значения False предотвращает изменение пользовательского форматирования со стороны приложения. В противном случае Excel может удалить полужирный шрифт из рядов с суммарными данными.

Как видно из рис. 18.4, требуется создать следующие группы:

- ряд 2-4;
- ряд 7-9;
- ряд 2-11.

Для этого используется метод Group объекта Range. Вот код, который выполняет разбиение на группы, показанные на рис. 18.4:

```
With ActiveSheet
    .Rows("2:4").Group
    .Rows("7:9").Group
    .Rows("2:11").Group
End With
```

Положением кнопок, раскрывающих/сворачивающих столбцы с данными при группировке, управляет свойство SummaryColumn объекта Outline. Щелчок на кнопке с номером x раскрывает все уровни, расположенные сверху, а все уровни, расположенные ниже, включая уровень x, сворачивает. Соответственно, все уровни на уровне x и выше будут видимыми, а расположенные ниже — скрытыми.

Эту задачу можно выполнить и с помощью метода ShowLevels объекта Outline. Вот его синтаксис:

```
OutlineObject.ShowLevels(RowLevels, ColumnLevels)
```

Например, выполнение кода:

```
ActiveSheet.Outline.ShowLevels 2
```

эквивалентно щелчку на кнопках, помеченных цифрой 2, и делает видимыми только расположенные выше уровни, как показано на рис. 18.5.

1	2	3		A	B	C	D
			1		Col1	Col2	Col3
	+		5	SubTotal1	432	89	590
	.		6				
	+		10	SubTotal2	579	46	511
	.		11				
	-		12	Total	1011	135	1101

Рис. 18.5. Свернутое до второго уровня схематичное представление

- Объект PageSetup представляет собой форматирование каждого листа (например, поля и размер страницы). Каждый из параметров форматирования настраивается заданием соответствующего свойства объекта PageSetup.

Свойство PageSetup объекта Worksheet возвращает объект PageSetup соответствующего листа.

Свойства и методы объекта PageSetup перечислены в табл. 18.3. Все перечисленные элементы являются свойствами, за исключением метода PrintQuality.

Таблица 18.3. Свойства и методы объекта PageSetup

Application	FitToPagesWide	PrintErrors<v10>
BlackAndWhite	FooterMargin	PrintGridlines
BottomMargin	HeaderMargin	PrintHeadings
CenterFooter	LeftFooter	PrintNotes
CenterFooterPicture<v10>	LeftFooterPicture<v10>	PrintQuality
CenterHeader	LeftHeader	PrintTitleColumns
CenterHeaderPicture<v10>	LeftHeaderPicture<v10>	PrintTitleRows
CenterHorizontally	LeftMargin	RightFooter
CenterVertically	Order	RightFooterPicture<v10>
ChartSize	Orientation	RightHeader
Creator	PaperSize	RightHeaderPicture<v10>
Draft	Parent	RightMargin
FirstPageNumber	PrintArea	TopMargin
FitToPagesTall	PrintComments	Zoom

Для иллюстрации рассмотрим код, который определяет ряд свойств активного листа:

```
With ActiveSheet.PageSetup
    .LeftMargin = Application.InchesToPoints(1)
```



```
.RightMargin = Application.InchesToPoints(1)
.PrintTitleRows = "A1"
.PaperSize = xlPaperLetter
End With
```

Обратите внимание на функцию `InchesToPoints`, которая требуется для перевода дюймов в точки, так как большинство форматирующих свойств измеряются именно в точках. Согласно рис. 18.6, свойство `PrintTitleRows` приведет к тому, что слово `Report`, расположенное в ячейке A1, появится на каждой выводимой на печать странице.

	A	B	C	D
1	Report			
2	1			
3	2			
4	3			
5	4			
6	5			
7	6			

Рис. 18.6. Лист и свойство `PrintTitleRows`

Функции защиты в Excel XP

В Excel XP существует ряд дополнительных функций для защиты листов, начиная с объекта `Protection`, о котором мы поговорим ниже.

Объект `Protection`

При защите листа приложение Excel позволяет пользователю определить операции, которые будут допустимы в незаблокированных ячейках. На пользовательском уровне это выполняется в диалоговом окне `Protection` (Защита листа). На уровне программирования для выполнения данной задачи используются свойства объекта `Protection`. Назначение большинства из этих свойств понятно по их названиям. Обратите внимание на то, что все они, кроме свойства `AllowEditRanges`, являются логическими. Вот их список:

- `AllowDeletingColumns`
- `AllowDeletingRows`
- `AllowEditRanges`
- `AllowFiltering`
- `AllowFormattingCells`
- `AllowFormattingColumns`
- `AllowFormattingRows`
- `AllowInsertingColumns`

- AllowInsertingHyperlinks
- AllowInsertingRows
- AllowSorting
- AllowUsingPivotTables

Например, если свойство AllowSorting имеет значение True, пользователь имеет возможность сортировать незаблокированные ячейки защищенного листа.

Свойство AllowEditRanges возвращает объект AllowEditRanges, о котором мы поговорим чуть позже.

Объект AllowEditRanges

Объект AllowEditRange позволяет защитить от редактирования определенный диапазон ячеек листа. После защиты диапазона таким способом и защиты всего листа редактирование ячеек потребует ввода пароля.

Ниже дан пример кода, защищающего паролем диапазон ячеек активного листа. Он также демонстрирует работу с семейством AllowEditRanges:

```
Sub ProtectRange()

Dim ws As Worksheet
Dim i As Integer

Set ws = Application.ActiveSheet

' Отключение защиты
ws.Unprotect

' Удаление всех диапазонов, защищенных в данный момент
'MsgBox ws.Protection.AllowEditRanges.Count
For i = 1 To ws.Protection.AllowEditRanges.Count
    Debug.Print ws.Protection.AllowEditRanges(i).Title
    ws.Protection.AllowEditRanges(i).Delete
Next

' Создание защищенного диапазона
ws.Protection.AllowEditRanges.Add _
    Title:="Headings", _
    Range:=Range("A1:A4"), _
    Password:="hide"

' Защита листа (иначе защита диапазона не будет включена)
ws.Protect

End Sub
```

Свойства объекта AllowEditRange:

- Свойство Range возвращает или задает диапазон, связанный с объектом AllowEditRange.
- Свойство Title возвращает или задает заголовок (имя) диапазона, связанного с объектом AllowEditRange.
- Свойство Users возвращает семейство UserAccessObjects, связанное с объектом AllowEditRange. Подробную информацию о нем вы найдете в следующем разделе.

Методы объекта AllowEditRange:

- Метод ChangePassword изменяет пароль, связанный с объектом AllowEditRange.
- Метод Delete удаляет объект AllowEditRange.
- Метод Unprotect снимает защиту с книги.

Объект UserAccess

Объект UserAccess дает возможность определенным пользователям редактировать защищенный диапазон без ввода пароля. Например, обратите внимание на код, обеспечивающий доступ к редактированию защищенного диапазона для пользовательского имени steve:

```
Sub AddUser()
```

```
Dim ws As Worksheet  
Dim ua As UserAccess
```

```
Set ws = Application.ActiveSheet
```

```
' ПРИМЕЧАНИЕ: Этот код работает только после снятия защиты с листа!  
ws.Unprotect  
Set ua = ws.Protection.AllowEditRanges(1).Users.Add("steve", True)
```

```
End Sub
```

Помните, что код работает без ошибок только после снятия защиты с листа.

Объект UserAccess имеет логическое свойство AllowEdit, свойство Name, предназначенное только для чтения, и метод Delete.

Семейство UserAccessList содержит текущие объекты UserAccess.

Пример: вывод на печать листов

Добавим функцию PrintSheets к надстройке SRXUtils. На данный момент служебная программа Print находится в надстройке Print.utl, и ее функция сводится к показу информационного окна. Для начала она должна вызывать диалоговое окно, показанное на рис. 18.7, со списком всех листов активной книги. Пользователю достаточно выделить один или несколько листов и щелкнуть на кнопке Печать, чтобы вывести на печать нужную информацию.



Рис. 18.7. Диалоговое окно Печать листов

Процесс разработки служебной программы для распечатки листов состоит из следующих этапов. Для начала требуется открыть книгу Print.xls. После

внесения необходимых изменений ее нужно сохранить под именем Print.utl. Если эта служебная программа уже загружена, потребуется отключить надстройку SRXUtils.xla (если она загружена) и закрыть книгу SRXUtils.xls (если она открыта).

Создание объекта UserForm

В книге Print.xls создайте диалоговое окно, показанное на рис. 18.7. Присвойте ему имя dlgPrintSheets, а в поле Caption введите название «Распечатка листов». Затем создайте процедуру PrintSheets:

```
Public Sub PrintSheets()  
    dlgPrintSheets.Show  
End Sub
```

Диалоговое окно dlgPrintSheets имеет две командные кнопки и одно поле со списком.

Поле со списком

Выделите поле со списком в форме, показанной на рис. 18.7. В окне Properties укажите свойства, перечисленные в табл. 18.4.

Таблица 18.4. Свойства поля со списком

Свойство	Значение
Name	LstSheets
TabIndex	0
MultiSelect	FrmMultiSelectExtended

Свойство MultiSelect имеет значение frmMultiSelectExtended, то есть пользователь может с помощью клавиши Ctrl выделить несколько непоследовательных имен листов, а с помощью клавиши Shift — набор последовательных имен листов.

Свойство TabIndex указывает не только последовательность активизации элементов формы при нажатии клавиши Tab, но и элемент, который будет активен при первой загрузке формы. Так как изначально активным должно быть поле со списком, свойству TabIndex этого элемента присваивается значение 0.

Кнопка Печать

Выделите командную кнопку Печать и укажите свойства, перечисленные в табл. 18.5, в окне Properties.

Таблица 18.5. Свойства кнопки Печать

Свойство	Значение
Name	cmdPrint
Accelerator	P
Caption	Печать
TabIndex	1

Кнопка Отмена

Выделите командную кнопку Отмена и укажите свойства, перечисленные в табл. 18.6, в окне Properties.

Таблица 18.6. Свойства кнопки Отмена

Свойство	Значение
Name	cmdCancel
Accelerator	C
Caption	Отмена
TabIndex	2
Cancel	True

Если свойству Cancel кнопки cmdCancel присвоено значение True, нажатие кнопки происходит, когда пользователь нажимает клавишу Esc. То есть нажатие этой клавиши будет приводить к исчезновению диалогового окна.

Создание кода для объекта UserForm

Пришло время связать с элементами управления некий код.

Область описаний

Область описаний объекта dlgPrintSheets должна содержать описание переменных уровня модуля, как показано в листинге 18.1.

Листинг 18.1. Описание переменных уровня модуля

```
Option Explicit
Dim cSheets As Integer
Dim sSheetNames() As String
```

Код кнопки Отмена

Код, соответствующий кнопке Отмена, показан в листинге 18.2.

Листинг 18.2. Обработка события cmdCancel_Click

```
Private Sub cmdCancel_Click()
    Unload Me
End Sub
```

Код кнопки Печать

Щелчок на кнопке Печать приводит к распечатке выделенных листов и выгрузке формы. Соответствующий код показан в листинге 18.3.

Листинг 18.3. Обработка события cmdPrint_Click

```
Private Sub cmdPrint_Click()
    PrintSelectedSheets
    Unload Me
End Sub
```

Событие Initialize формы

Событие Initialize объекта UserForm должно приводить к появлению в поле со списком листов активной книги. В нашем приложении используется массив уровня

модуля `sSheetNames`, в котором будут находиться названия листов, и целая переменная `cSheets`, хранящая количество листов. Они были определены в листинге 18.1. Эти параметры нужно связать с событием `Initialize`, а затем использовать массив для заполнения поля со списком, как показано в листинге 18.4. Данные переменные будут фигурировать и в процедуре вывода на печать, именно поэтому они были определены на уровне модуля.

Обратите внимание на использование оператора `ReDim` для изменения размерности массива. Это требуется, так как невозможно узнать заранее, сколько листов содержится в книге.

Листинг 18.4. Процедура для события `Initialize` объекта `UserForm`

```
Private Sub UserForm_Initialize()

Dim ws As Object      'Рабочий лист
ReDim sSheetNames(1 To 10)

lstSheets.Clear
cSheets = 0
For Each ws In ActiveWorkbook.Sheets

    cSheets = cSheets + 1

    ' При необходимости изменим размерность массива
    If UBound(sSheetNames) < cSheets Then
        ReDim Preserve sSheetNames(1 To cSheets + 5)
    End If

    ' Сохранение имен листов
    sSheetNames(cSheets) = ws.Name

    ' Добавление имен листов в поле со списком
    lstSheets.AddItem sSheetNames(cSheets)

Next
End Sub
```

Процедура `PrintSheets`

Основная процедура служебной программы по выводу на печать листов показана в листинге 18.5. Обратите внимание на два момента. Во-первых, в книге вообще может не содержаться листов. Во-вторых, пользователь может щелкнуть на кнопке `Печать`, не выделив ни одного листа в поле со списком.

Важно понимать, что все поля со списком представляют собой массивы, нумерация которых начинается с нуля, а нумерация нашего массива со списком листов книги начинается с единицы. То есть следует помнить, что элементу массива с номером i соответствует индекс $i + 1$.

Листинг 18.5. Процедура `PrintSelectedSheets`

```
Sub PrintSelectedSheets()

Dim i As Integer
Dim bNoneSelected As Boolean

bNoneSelected = True

If cSheets = 0 Then
```



```
MsgBox "В этой книге нет ни одного листа!", vbExclamation
Exit Sub
Else
  For i = 0 To 1stSheets.ListCount - 1
    If 1stSheets.Selected(i) Then
      bNoneSelected = False
      ' Нумерация поля со списком начинается с 0, а массива – с 1
      ActiveWorkbook.Sheets(sSheetNames(i + 1)).PrintOut
    End If
  Next
End If

If bNoneSelected Then
  MsgBox "Вы не выделили ни одного листа в списке!", vbExclamation
End If

End Sub
```


19 **Объект Range**

Объект Range является одним из чаще всего используемых представителей объектной модели Excel. Ведь для работы с частью листа Excel сначала требуется определить ее как объект Range.

Как написано в справочной документации Microsoft: «Объект Range представляет собой ячейку, строку, столбец, выделенные наборы ячеек, расположенные одним или несколькими блоками, или даже трехмерный диапазон».

В табл. 19.1 перечислены 158 свойств и методов объекта Range.

Таблица 19.1. Свойства и методы объекта Range

_Default	End	Phonetic
_NewEnum	EntireColumn	Phonetics<v9>
_PasteSpecial<v10>	EntireRow	PivotCell<v10>
_PrintOut<v9>	Errors<v10>	PivotField
Activate	FillDown	PivotItem
AddComment	FillLeft	PivotTable
AddIndent	FillRight	Precedents
Address	FillUp	PrefixCharacter
AddressLocal	Find	Previous
AdvancedFilter	FindNext	PrintOut
AllowEdit<v10>	FindPrevious	PrintPreview
Application	Font	QueryTable
ApplyNames	FormatConditions	Range
ApplyOutlineStyles	Formula	ReadingOrder
Areas	FormulaArray	RemoveSubtotal
AutoComplete	FormulaHidden	Replace
AutoFill	FormulaLabel	Resize
AutoFilter	FormulaLocal	Row
AutoFit	FormulaR1C1	RowDifferences
AutoFormat	FormulaR1C1Local	RowHeight
AutoOutline	FunctionWizard	Rows
BorderAround	GoalSeek	Run
Borders	Group	Select
Calculate	HasArray	SetPhonetic<v9>
Cells	HasFormula	Show

Characters	Height	ShowDependents
CheckSpelling	Hidden	ShowDetail
Clear	HorizontalAlignment	ShowErrors
ClearComments	Hyperlinks	ShowPrecedents
ClearContents	ID<v9>	ShrinkToFit
ClearFormats	IndentLevel	SmartTags<v10>
ClearNotes	Insert	Sort
ClearOutline	InsertIndent	SortSpecial
Column	Interior	SoundNote
ColumnDifferences	Item	Speak<v10>
Columns	Justify	SpecialCells
ColumnWidth	Left	Style
Comment	ListHeaderRows	SubscribeTo
Consolidate	ListNames	Subtotal
Copy	LocationInTable	Summary
CopyFromRecordset	Locked	Table
CopyPicture	Merge	Text
Count	MergeArea	TextToColumns
CreateNames	MergeCells	Top
CreatePublisher	Name	Ungroup
Creator	NavigateArrow	UnMerge
CurrentArray	Next	UseStandardHeight
CurrentRegion	NoteText	UseStandardWidth
Cut	NumberFormat	Validation
DataSet	NumberFormatLocal	Value
Delete	Offset	Value2
Dependents	Orientation	VerticalAlignment
DialogBox	OutlineLevel	Width
DirectDependents	PageBreak	Worksheet
DirectPrecedents	Parent	WrapText
Dirty<v10>	Parse	
EditionOptions	PasteSpecial	

В настоящей главе сначала будут рассмотрены способы определения объекта Range, затем поговорим о ряде его свойств и методов, перечисленных в табл. 19.1. Как уже упоминалось, в этой книге не ставится цель рассказать обо всех аспектах объектной модели Excel. Наша задача — дать основные сведения по данной теме и основу для дальнейшего самостоятельного изучения материала.

Объект Range как семейство

Необычность объекта Range состоит в том, что он может рассматриваться и как обычный объект, и как семейство. Например, он включает метод Item и свойство Count. С другой стороны, у него имеется множество свойств и методов, которые отсутствуют у семейств. В частности, среднее число свойств и методов у других семейств составляет 19, в то время как объект Range имеет 158 свойств и методов.

В действительности объект Range следует рассматривать как семейство, содержащее остальные объекты Range. Для иллюстрации рассмотрим следующий код:

```
Dim rng as Range
Set rng = Range("A1", "C5").Cells      ' Показывает 15
MsgBox rng.Count
Set rng = Range("A1", "C5").Rows      ' Показывает 5
MsgBox rng.Count
Set rng = Range("A1", "C5").Columns    ' Показывает 3
MsgBox rng.Count
```

Этот код присваивает переменной rng совокупность всех ячеек, строк и столбцов в диапазоне A1:C5. В каждом случае функция MsgBox показывает точное количество элементов семейства. Еще раз обращаем ваше внимание на то, что в объектной модели Excel отсутствуют такие объекты, как ячейка, строка или столбец. Вместо них используется объект Range. Поэтому переменная rng принадлежит именно к этому типу.

Если не указать в явном виде тип объекта Range, приложение интерпретирует его как набор ячеек. Для иллюстрации рассмотрим код:

```
Dim rng As Range
Set rng = Range("A1", "C5")
MsgBox rng.Count
MsgBox rng(6).Value      ' Развертка массива по строкам
```

Он отображает количество ячеек в заданном диапазоне, а также значение шестой ячейки диапазона, сосчитанное при развертке массива по строкам. То есть отсчет начинается с первого ряда и ведется слева направо. А код:

```
Dim rng As Range
Dim oCell As Range
Set rng = Range("A1", "C5")
For Each oCell In rng
    Debug.Print oCell.Value
Next
```

будет проверять каждую ячейку диапазона rng, отображая ее значение в окне Immediate.

Определение объекта Range

В объектной модели Excel существуют 113 свойств и методов, которые возвращают объект Range. Если посчитать только *отличающиеся* друг от друга названия, количество этих свойств и методов снизится до 51. Они перечислены в табл. 19.2. Например, свойство BottomRightCell является принадлежностью 21 различного объекта, как и свойство TopLeftCell.

Таблица 19.2. Свойства и методы, возвращающие объект Range

_Default	End	Range
ActiveCell	EntireColumn	RangeSelection
BottomRightCell	EntireRow	RefersToRange
Cells	Find	Resize
ChangingCells	FindNext	ResultRange
CircularReference	FindPrevious	RowDifferences
ColumnDifferences	GetPivotData	RowRange

ColumnRange	Intersect	Rows
Columns	Item	SourceRange
CurrentArray	LabelRange	SpecialCells
CurrentRegion	Location	TableRange1
DataBodyRange	MergeArea	TableRange2
DataLabelRange	Next	ThisCell
DataRange	Offset	TopLeftCell
Dependents	PageRange	Union
Destination	PageRangeCells	UsedRange
DirectDependents	Precedents	VisibleRange
DirectPrecedents	Previous	

Рассмотрим наиболее распространенные способы определения объекта Range.

Свойство Range

Свойство Range применяется к объектам Application, Range и Worksheet. Обратите внимание на то, что записи

Application.Range

и

ActiveSheet.Range

эквивалентны.

Если свойство Range упоминается без уточнений внутри модуля кода листа, значит, оно применяется именно к этому листу. Если же это свойство без уточнений упоминается внутри модуля кода книги, оно применяется к активному листу этой книги.

Соответственно, если в модуле кода для листа Sheet2 появляется вот такая запись

Worksheets(1).Activate
Range("D1").Value = "Тест".

то этот модуль сначала активизирует лист Sheet1, но все равно помещает слово "Тест" в ячейку D1 листа Sheet2.

Свойство Range имеет два варианта синтаксиса. Первый вариант выглядит так:

object.Range(Name).

где аргумент Name представляет собой имя диапазона. Это должна быть ссылка в стиле A1, причем имя может включать оператор диапазона (двоеточие), оператор объединения (запятая) или оператор пересечения (пробел). Знак доллара (\$) в составе параметра Name игнорируется. Кроме того, можно использовать название именованного диапазона.

Для иллюстрации рассмотрим примеры:

Range("A2")
Range("A2:B3")
Range("A2:F3 A1:D5") ' Пересечение
Range("A2:F3, A1:D5") ' Объединение

Разумеется, для преобразования формулы в стиле R1C1 в формулу в стиле A1 можно использовать метод ConvertFormula и только после этого применять свойство Range:

Range(Application.ConvertFormula("R2C5:R6C9", x1R1C1, x1A1))

И наконец, чтобы вернуть диапазон с именем `TestRange`, можно написать:

```
Range(Application.Names("TestRange"))
```

или

```
Range(Application.Names!TestRange)
```

Второй вариант синтаксиса свойства `Range` выглядит следующим образом:

```
object.Range(Cell1, Cell2)
```

Здесь *Cell1* — ячейка в верхнем левом углу диапазона, а *Cell2* — ячейка в его нижнем правом углу. Например:

```
Range("D4", "F8")
```

Кроме того, параметры *Cell1* и *Cell2* могут быть объектами `Range`, а именно строкой или столбцом. Например, следующая строка возвращает объект `Range`, который представляет собой вторую и третью строки активного листа:

```
Range(Rows(2), Rows(3))
```

Когда свойство `Range` применяется к объекту `Range`, все ссылки относятся к левой верхней угловой ячейке рассматриваемого диапазона. Например, если параметр *rng* представляет второй столбец активного листа, то запись:

```
rng.Range("A2")
```

соответствует второй ячейке в этом столбце, а вовсе не ячейке A2 листа. Кроме того, выражение:

```
rng.Range("B2")
```

представляет ячейку C2, так как именно она находится во втором столбце и второй строке от ячейки B1, которая располагается в верхнем левом углу диапазона *rng*.

Свойство Cells

В объектной модели Excel нет ни семейства `Cells`, ни объекта `Cell`. Существующее свойство `Cells` возвращает объект `Range`. Например, следующая строка кода возвращает число 8:

```
Range("A1:B4").Cells.Count
```

Кстати, строка `Cells.Count` возвращает значение 16 777 216, то есть $256 \times 65\,536$.

Свойство `Cells` применимо к объектам `Application`, `Range` и `Worksheet`. Кроме того, оно является глобальным. Будучи применено к объекту `Worksheet`, оно возвращает объект `Range`, который состоит из всех ячеек листа. Более того, следующие записи полностью эквивалентны:

```
Cells
```

```
Application.Cells
```

```
ActiveSheet.Cells
```

Примененное к объекту `Range` свойство `Cells` возвращает этот объект обратно, то есть не производит никакого действия.

Запись:

```
Cells(i,j)
```

возвращает объект `Range`, представляющий собой ячейку на пересечении *i*-строки и *j*-столбца. Поэтому строки:

```
Cells(1,1)
```


и

```
Range("A1")
```

эквивалентны.

Преимущество свойства Cells над методом Range состоит в том, что оно допускает целочисленные переменные. Например, следующий код ищет в первых ста строках четвертого столбца первую ячейку, содержащую слово "Тест". При нахождении такой ячейки она выделяется. В противном случае появляется информационное окно с сообщением.

```
Dim r As Long
For r = 1 To 100
    If Cells(r, 4).Value = "Тест" Then
        Cells(r, 4).Select
        Exit For
    End If
Next
If r = 101 then MsgBox "Ячейка отсутствует."
```

Существует также возможность скомбинировать свойства Cells и Range. Например, рассмотрим код:

```
Dim r As Long
Dim rng As Range
With ActiveSheet
    For r = 1 To 100
        If Cells(r, r).Value <> "" Then
            Set rng = .Range(.Cells(1, 1), .Cells(r, r))
            Exit For
        End If
    Next
End With
rng.Select
```

Этот код перебирает диагональные ячейки (номер строки которых равен номеру столбца), пока не будет найдена непустая ячейка. Затем переменной rng присваивается диапазон в виде прямоугольника, верхний левый угол которого находится в ячейке A1, а нижний правый — в ячейке, найденной в процессе поиска.

Свойства Column, Row и Rows

В объектной модели Excel отсутствуют семейства Columns и Rows. При этом свойство Columns возвращает набор объектов Range, каждый из которых представляет собой столбец. Соответственно, код:

```
ActiveSheet.Columns(i)
```

представляет собой объект Range, который относится к i-столбцу активного листа, а также к набору ячеек в этом столбце. Точно так же код:

```
ActiveSheet.Rows(i)
```

относится к i-строке активного листа.

Свойства Columns и Rows также можно использовать вместе с объектом Range. Пожалуй, объект, описываемый выражением

```
rng.Columns
```

проще всего рассмотреть как набор всех столбцов листа, *перенумерованных* таким образом, что первым является крайний слева столбец, пересекающий диапазон *rng*.

Для иллюстрации рассмотрим код, результат выполнения которого показан на рис. 19.1:

```
Dim i As Integer
Dim rng As Range
Set rng = Range("D1:E1, G1:I1")
rng.Select
MsgBox "Первый столбец в диапазоне " & rng.Column ' Показано 4
MsgBox "Число столбцов " & rng.Columns.Count ' Показано 2
For i = -(rng.Column - 2) To rng.Columns.Count + 1
    rng.Columns(i).Cells(1, 1).Value = i
Next
```

	A	B	C	D	E	F	G	H	I
1	-2	-1	0	1	2	3			
2									

Рис. 19.1. Прерывающийся диапазон

Обратите внимание на то, как диапазон *rng* выделен на рис. 19.1. В него входит и ячейка D1. Свойство *Column* объекта *Range* возвращает крайний слева столбец, пересекающий рассматриваемый диапазон. В свою очередь свойство *Row* возвращает самую верхнюю строку, пересекающую этот диапазон. Однако первое информационное окно покажет значение 4.

Теперь с точки зрения диапазона *rng* *Column(1)* — это столбец номер 4 активного листа, то есть столбец D. Соответственно, *Column(0)* — это столбец номер 3, то есть столбец C, который не является частью диапазона *rng*. В действительности первый столбец листа имеет номер:

```
-(rng.Column - 2)
```

Именно с этого значения был начат цикл *For*.

Теперь обратите внимание на тот факт, что строка

```
rng.Columns.Count
```

возвращает значение 2. Именно такое значение отобразится во втором информационном окне. Это немного неожиданно. Но по каким-то причинам компания Microsoft сделала так, что свойство *Count* в записи *rng.Columns* возвращает число столбцов, пересекающих только крайнюю левую сторону диапазона. В данном случае это область D1:E1. Наконец, обратим внимание на тот факт, что строка:

```
rng.Columns(3)
```

возвращает столбец F, не принадлежащий к рассматриваемому диапазону.

В качестве еще одной иллюстрации рассмотрим диапазон, показанный на рис. 19.2. Это объединенные области B4:C5 и E2:E7.

Код:

```
Dim rng As Range
Set rng = Range("B4:C5, E2:E7")
MsgBox rng.Columns(1).Cells(1, 1).Value
```

отображает информационное окно, содержащее букву x, которая находится в ячейке B4. Это происходит потому, что индексы в свойстве *Cells* отсчитываются относительно верхней ячейки в крайней левой области диапазона.

	A	B	C	D	E	F
1						
2						
3						
4		x				
5						
6						
7						
8						
9						

Рис. 19.2. Диапазон, объединяющий две области

Для обозначения столбцов можно использовать как целые числа, так и буквы в двойных кавычках. Например:

```
Columns(5)
```

или:

```
Columns("E")
```

Можно также написать:

```
Columns("A:D").
```

чтобы указать на столбцы от A до D. Другой вариант указания на несколько столбцов одновременно выглядит так:

```
Rows("1:3")
```

Так как запись вида:

```
Columns("C:D", "G:H")
```

недопустима, вместо нее используется комбинация методов Union, Columns и Rows. Например, код:

```
Dim rng As Range
Set rng = Union(Rows(3), Rows(5), Rows(7))
rng.Select
```

выделяет третью, пятую и седьмую строки листа, в модуле которого расположен данный код, или же активного листа, если код находится в модуле книги или в стандартном модуле.

Свойство Offset

Свойство Offset возвращает диапазон, отстоящий от указанного пользователем на определенное число строк и/или столбцов. Вот его синтаксис:

```
RangeObject.Offset(RowOffset, ColumnOffset).
```

где параметр *RowOffset* представляет собой количество строк, а параметр *ColumnOffset* — число столбцов, на которое смещен диапазон. Оба этих параметра являются необязательными и по умолчанию имеют значение 0. Кроме того, оба они могут принимать как положительные, так и отрицательные значения.

Например, следующий код ищет пустую ячейку в ста ячейках, расположенных непосредственно справа от ячейки D2:

```
Dim rng As Range
Dim i As Integer
```



```
Set rng = Range("D2")
For i = 1 To 100
    If rng.Offset(0, i).Value = "" Then
        MsgBox "Найдена пустая ячейка на расстоянии " & i & " от ячейки D2"
    End If
Next
```

Дополнительные свойства и методы объекта Range

Теперь рассмотрим дополнительные свойства и методы объекта Range.

Метод Activate

Метод Activate активизирует (или выделяет) диапазон, в котором он используется. Он применим не только к объекту Range, но и к ряду других объектов, например Window, Worksheet или Workbook.

Метод AddComment

Метод AddComment добавляет объект Comment в диапазон, состоящий из одной ячейки. Вот его синтаксис:

```
RangeObject.AddComment(Text).
```

где параметр *Text* представляет собой текст комментария. Например, код:

```
Dim rng As Range
Dim c As Comment
Set rng = Range("B2")
Set c = rng.AddComment("Это комментарий")
```

добавляет комментарий с текстом «Это комментарий» в ячейку B2. Если объект RangeObject состоит из нескольких ячеек, появляется сообщение об ошибке.

Свойство Address (только чтение, тип String)

Свойство Address возвращает диапазон ссылок объекта Range в виде строки. Вот его синтаксис:

```
RangeObject.Address(RowAbsolute, ColumnAbsolute, _
    ReferenceStyle, External, RelativeTo)
```

Параметр *RowAbsolute* по умолчанию имеет значение True и возвращает строковую часть ссылки в абсолютной форме. Параметр *ColumnAbsolute* также по умолчанию имеет значение True и возвращает столбцовую часть ссылки в абсолютном виде.

Параметр *ReferenceStyle* может принимать значения констант из перечисления *XlReferenceStyle*, то есть иметь значения *xlA1* (по умолчанию) или *xlR1C1*.

В результате присваивания параметру *External* значения True возвращаются внешние ссылки, то есть ссылки, действительные вне рассматриваемого листа. По умолчанию этот параметр имеет значение False, что позволяет возвращать ссылки, действительные только для данного листа.

И наконец, параметр *RelativeTo* используется только в том случае, если параметры *RowAbsolute* и *ColumnAbsolute* имеют значение *False*, а параметр *ReferenceStyle* — значение *xlA1*. При этом требуется указать точку ссылки (ячейку), которая будет использоваться как относительный адрес.

Для иллюстрации действия свойства *Address* рассмотрим пример:

```
Set rng = Range("B2")
rng.Address(ReferenceStyle:=xlA1)           ' Возвращает $B$2
rng.Address(ReferenceStyle:=xlA1, _
    External:=True)                         ' Возвращает [Book1]Sheet1!$B$2
rng.Address(ReferenceStyle:=xlR1C1)         ' Возвращает R2C2
rng.Address(RowAbsolute:=False, _
    ColumnAbsolute:=False, ReferenceStyle:=xlA1) ' Возвращает B2
rng.Address(RowAbsolute:=False, _
    ColumnAbsolute:=False, ReferenceStyle:=xlR1C1, _
    RelativeTo:=Range("D1"))                ' Возвращает R[1]C[-2]

Set rng = Range("B2:D5")
rng.Address(ReferenceStyle:=xlA1)           ' Возвращает $B$2:$D$5
rng.Address(ReferenceStyle:=xlR1C1)         ' Возвращает R2C2:R5C4
rng.Address(RowAbsolute:=False, _
    ColumnAbsolute:=False, ReferenceStyle:=xlA1) ' Возвращает B2:D5
rng.Address(RowAbsolute:=False, _
    ColumnAbsolute:=False, ReferenceStyle:=xlR1C1, _
    RelativeTo:=Range("D1"))                ' Возвращает R[1]C[-2]:R[4]C
```

Метод AutoFill

Метод *AutoFill* управляет автозаполнением ячеек диапазона. Вот его синтаксис:

RangeObject.AutoFill(Destination, Type)

Параметр *Destination* представляет собой объект *Range*, ячейки которого требуется заполнить. Он должен включать и ячейки с образцом данных, по которому будет производиться автозаполнение.

Необязательный параметр *Type* указывает тип заполнения. Он может принимать значения констант из следующего перечисления:

```
Enum XlAutoFillType
    xlFillDefault = 0
    xlFillCopy = 1
    xlFillSeries = 2
    xlFillFormats = 3
    xlFillValues = 4
    xlFillDays = 5
    xlFillWeekdays = 6
    xlFillMonths = 7
    xlFillYears = 8
    xlLinearTrend = 9
    xlGrowthTrend = 10
End Enum
```

В документации по Excel в данном случае дана ссылка на несуществующее перечисление *XlFillType*.

Если аргумент *Type* имеет значение *xlFillDefault* или вообще пропущен, приложение попытается выбрать наиболее подходящий способ заполнения на основе уже введенных данных.

Для иллюстрации рассмотрим код:

```
Range("A1:B1").AutoFill Range("A1:K1"),
```

который заполняет диапазон ячеек C1:K1 на основе данных, введенных в ячейки диапазона A1:B1. Если ячейка A1 содержит значение 1, а ячейка B1 — значение 2, остальные ячейки будут заполнены возрастающими на единицу целыми числами, начиная с трех. Обратите внимание на то, что ячейки A1 и B1 включены в диапазон назначения.

В качестве дополнительной иллюстрации рассмотрим лист, показанный на рис. 19.3, в котором ячейка B1 содержит формулу =A1*A1.

	A	B
1	3	9
2	4	
3	5	
4	6	
5	7	
6		

Рис. 19.3. Лист, в котором требуется заполнить диапазон B1:B5

Результат выполнения кода:

```
Range("B1").AutoFill Range("B1:B5")
```

показан на рис. 19.4.

	A	B
1	3	9
2	4	16
3	5	25
4	6	36
5	7	49

Рис. 19.4. Результат автозаполнения диапазона B1:B5

Следует упомянуть о потенциальной проблеме при использовании метода AutoFill. При выполнении кода формула из ячейки-источника копируется в заполняемые ячейки с соответствующими изменениями, в то время как значение из ячейки-источника копируется без изменений. Соответственно, формулы в заполненных ячейках будут верными, а значения — нет. Решить эту проблему можно с помощью метода Calculation.

Метод AutoFilter

Метод AutoFilter имеет два варианта синтаксиса, соответствующих двум различным функциям. Синтаксис:

```
RangeObject.AutoFilter
```

просто формирует раскрывающиеся списки для входящих в диапазон столбцов.

Если же написать:

```
RangeObject.AutoFilter(Field, Criteria1, Operator, Criteria2)
```

появятся раскрывающиеся списки и фильтры, вызываемые с помощью функции AutoFilter (Автофильтр).

Необязательный параметр *Field* показывает смещение поля, на основе которого производится фильтрация. Это целое число, отсчитываемое слева. Соответственно, крайнее слева поле считается полем номер один.

Необязательный параметр *Criteria1* — это строка, указывающая критерий фильтрации. В данном случае можно использовать знак равенства (“=”) для поиска пустых полей и знак “<>” для поиска заполненных. При пропуске данного аргумента в качестве критерия фигурирует значение All.

Если параметр *Operator* имеет значение xlTop10Items, а параметр *Criteria1* указывает, что фильтрация идет по целым числам, будет показан набор элементов, число которых не должно быть равным десяти.

Параметр *Operator* может принимать значения констант из следующего перечисления:

```
Enum XlAutoFilterOperator
    xlAnd = 1
    xlOr = 2
    xlTop10Items = 3
    xlBottom10Items = 4
    xlTop10Percent = 5
    xlBottom10Percent = 6
End Enum
```

Если этот параметр имеет значение xlAnd или xlOr, требуется задать составной критерий на основе параметров *Criteria1* и *Criteria2*.

Для иллюстрации рассмотрим лист, показанный на рис. 19.5.

	A	B
1	Number	Year
2	1	1998
3	2	1996
4	3	1994
5	4	1997
6	5	1998
7	6	1994
8	7	1993
9	8	1994
10	9	1997
11	10	1998

Рис. 19.5. Вид листа до фильтрации

Код

```
Range("A1:B5").AutoFilter 2, "1997", xlOr, "1998"
```

оставит видимыми в диапазоне A1:B5 только строки, соответствующие 1997 и 1998 годам. На остальные строки листа этот код не окажет никакого влияния. В результате будет получен лист, показанный на рис. 19.5, у которого пропущены третья и четвертая строки.

Напомним, что свойство `AutoFilterMode` объекта `Worksheet` имеет значение `True`, если на листе видны раскрывающиеся списки с параметрами фильтрации. Присвоив этому свойству значение `False`, можно убрать эти списки, но вызвать их появление, присвоив свойству `AutoFilterMode` значение `True`, невозможно.

Следует также помнить, что свойство `FilterMode` имеет значение `True`, если лист находится в режиме фильтрации. Соответственно, если, к примеру, на листе имеются раскрывающиеся списки с критериями фильтрации, но сама операция фильтрации не производится, свойство `AutoFilterMode` имеет значение `True`, в то время как свойство `FilterMode` имеет значение `False`. При начале фильтрации последнее свойство приобретает значение `True`.

Метод `AutoFit`

Метод `AutoFit` изменяет ширину столбцов и высоту строк (в зависимости от типа диапазона) в соответствии с размером содержимого ячеек. Вот его синтаксис:

`RangeObject.AutoFit`.

где параметр *RangeObject* относится к объекту `Range`, состоящему из одного или нескольких столбцов или из одной или нескольких строк (но не того и другого одновременно). В противном случае появляется сообщение об ошибке. Если диапазон состоит из столбцов, меняется их ширина, если из строк — их высота.

Метод `AutoFormat`

Метод `AutoFormat` позволяет осуществить автоматическое форматирование диапазона на основе заранее определенных параметров. Вот его синтаксис:

`RangeObject.AutoFormat(Format, Number, Font, _
Alignment, Border, Pattern, Width)`

Все параметры данного метода являются необязательными. Параметр *Format* может принимать значения констант из следующего перечисления:

```
Enum XlRangeAutoFormat
xlRangeAutoFormat3DEffects1 = 13
xlRangeAutoFormat3DEffects2 = 14
xlRangeAutoFormatAccounting1 = 4
xlRangeAutoFormatAccounting2 = 5
xlRangeAutoFormatAccounting3 = 6
xlRangeAutoFormatAccounting4 = 17
xlRangeAutoFormatClassic1 = 1
xlRangeAutoFormatClassic2 = 2
xlRangeAutoFormatClassic3 = 3
xlRangeAutoFormatClassicPivotTable = 31
xlRangeAutoFormatColor1 = 7
xlRangeAutoFormatColor2 = 8
xlRangeAutoFormatColor3 = 9
xlRangeAutoFormatList1 = 10
xlRangeAutoFormatList2 = 11
xlRangeAutoFormatList3 = 12
xlRangeAutoFormatLocalFormat1 = 15
xlRangeAutoFormatLocalFormat2 = 16
xlRangeAutoFormatLocalFormat3 = 19
xlRangeAutoFormatLocalFormat4 = 20
xlRangeAutoFormatNone = -4142
```



```

xlRangeAutoFormatPTNone = 42
xlRangeAutoFormatReport1 = 21
xlRangeAutoFormatReport10 = 30
xlRangeAutoFormatReport2 = 22
xlRangeAutoFormatReport3 = 23
xlRangeAutoFormatReport4 = 24
xlRangeAutoFormatReport5 = 25
xlRangeAutoFormatReport6 = 26
xlRangeAutoFormatReport7 = 27
xlRangeAutoFormatReport8 = 28
xlRangeAutoFormatReport9 = 29
xlRangeAutoFormatSimple = -4154
xlRangeAutoFormatTable1 = 32
xlRangeAutoFormatTable10 = 41
xlRangeAutoFormatTable2 = 33
xlRangeAutoFormatTable3 = 34
xlRangeAutoFormatTable4 = 35
xlRangeAutoFormatTable5 = 36
xlRangeAutoFormatTable6 = 37
xlRangeAutoFormatTable7 = 38
xlRangeAutoFormatTable8 = 39
xlRangeAutoFormatTable9 = 40
End Enum

```

Остальные параметры являются логическими и должны иметь заданное по умолчанию значение `True`. Только в этом случае в форматирование будут включены следующие параметры:

- *Number* — включает числовые форматы;
- *Font* — включает формат шрифтов;
- *Alignment* — включает выравнивание;
- *Border* — включает формат рамок;
- *Pattern* — включает формат рисунков;
- *Width* — включает ширину столбцов и высоту строк при автоформатировании.

Если диапазон включает в себя всего одну ячейку, метод `AutoFormat` форматирует и область, содержащую в себе эту ячейку. Свойство `CurrentRegion` будет рассмотрено немного позднее. Другими словами, две следующие записи эквивалентны:

```

Cells("A1").AutoFormat
Cells("A1").CurrentRegion.AutoFormat

```

Метод `BorderAround`

Метод `BorderAround` заключает диапазон в рамку, устанавливая при необходимости такие свойства рамки, как `Color`, `LineStyle` и `Weight`. Вот его синтаксис:

```
RangeObject.BorderAround(LineStyle, Weight, ColorIndex, Color)
```

Параметр `LineStyle` может принимать значения констант из следующего перечисления:

```

Enum XlLineStyle
xlLineStyleNone = -4142
xlDouble = -4119
xlDot = -4118
xlDash = -4115
xlContinuous = 1 ' Значение по умолчанию

```



```
xlDashDot = 4  
xlDashDotDot = 5  
xlSlantDashDot = 13  
End Enum
```

В документации Excel в данном случае упоминается несуществующее перечисление `xlBorderStyle`.

Необязательный параметр *Weight* задает толщину рамки и может принимать значения констант из следующего перечисления:

```
Enum XlBorderWeight  
xlMedium = -4138  
xlHairline = 1  
xlThin = 2 ' the default  
xlThick = 4  
End Enum
```

Свойство *Weight* используется только в том случае, когда параметр *LineStyle* пропущен или же имеет значение `xlContinuous`.

Необязательный параметр *ColorIndex* указывает цвет рамки в виде индекса в текущей цветовой палитре или в виде константы следующего перечисления:

```
Enum XlColorIndex  
xlColorIndexNone = -4142  
xlColorIndexAutomatic = -4105  
End Enum
```

Необязательный параметр *Color* также определяет цвет рамки в виде значения RGB. Имейте в виду, что указывать требуется только один из задающих цвет параметров.

Способ удаления рамок выглядит слегка неожиданно. Предположим, что рамка была создана с помощью кода:

```
rng.BorderAround LineStyle:=xlDash
```

Для удаления этой рамки можно воспользоваться кодом:

```
rng.BorderAround LineStyle:=xlLineStyleNone.
```

но это не даст никакого результата. Вместо этого нужно использовать код:

```
rng.Borders.LineStyle = xlLineStyleNone.
```

который отдельно удаляет рамку вокруг каждой ячейки диапазона.

Метод Calculate

Метод `Calculate`, применимый также к объектам `Workbook` и `Worksheet`, производит пересчет формул в ячейках указанного пользователем диапазона. Например, код:

```
Worksheets(1).Rows(2).Calculate
```

производит пересчет формул во всех ячейках второй строки первого листа.

Метод Clear

В Excel существует несколько вариантов метода возвращения в исходное состояние. Например, метод `Clear` удаляет все содержимое, формулы и форматирование из ячеек указанного пользователем диапазона. Метод `ClearContent` удаляет только содержимое ячеек (значения и/или формулы), но оставляет форматиро-

вание нетронутым. Если же требуется отменить форматирование, используется метод `ClearFormats`.

Методы `ColumnDifferences` и `RowDifferences`

Метод `ColumnDifferences` возвращает объект `Range`, представляющий собой все ячейки диапазона, содержимое которого отличается от содержимого сравнительных ячеек. Каждый столбец содержит одну такую ячейку. Вот синтаксис этого метода: `RangeObject.ColumnDifferences(ComparisonCell)`.

где параметр `ComparisonCell` задает диапазон, состоящий из одной ячейки. Этот параметр требуется только для распознавания строки, ячейка которой содержит сравнительное значение.

Для иллюстрации рассмотрим код, результат выполнения которого показан на рис. 19.6.

```
Dim rng As Range, rng2 As Range
Set rng = Range("A1:D6")
Set rng2 =
    rng.ColumnDifferences(Comparison:=Range("A1"))
rng2.Select
```

	A	B	C	D
1	x		y	a
2	x	x	x	b
3	y		x	c
4	x	y	x	d
5	x		y	a
6	x	x	x	e

Рис. 19.6. Результат применения метода `ColumnDifferences`

Так как параметром `ComparisonCell` в данном случае является ячейка `A1`, сравнительное значение находится в первой ячейке каждого столбца диапазона. Поэтому ячейки, не содержащие значения `x`, выделены в столбце `A`, непустые ячейки выделены в столбце `B`, ячейки, не содержащие значения `y`, выделены в столбце `C`, а ячейки, не содержащие значения `a`, выделены в столбце `D`.

Метод `RowDifferences` выполняет то же самое действие со строками листа.

Свойства `ColumnWidth` и `RowHeight`

Свойство `ColumnWidth` возвращает или задает ширину столбцов в указанном пользователем диапазоне. Возвращаемое значение измеряется в единицах, каждая из которых равна ширине одного символа, записанного стилем `Normal`. Для пропорциональных шрифтов используется ширина символа 0 (ноль).

Если ширина столбцов в диапазоне различается, свойство `ColumnWidth` возвращает значение `Null`.

Свойство `RowHeight` возвращает высоту всех рядов в диапазоне, измеряемую в точках. Если ряды диапазона имеют разную высоту, это свойство возвращает значение `Null`.

Свойства Width, Height, Top и Left

Свойства Width, Height, Top и Left возвращают соответствующие значения для всего диапазона, измеренные в точках. Например, свойство Top возвращает расстояние от верхней линии первой строки до верхней линии первой (крайней слева) области рассматриваемого диапазона.

Примененное к столбцу свойство Width возвращает ширину последнего. Однако соотношение между свойствами Width и ColumnWidth на первый взгляд выглядит несколько странно.

Например, рассмотрим следующий код:

```
Columns("A").ColumnWidth = 1
MsgBox Columns("A").Width      ' Результат 9.6
Columns("A").ColumnWidth = 2
MsgBox Columns("A").Width      ' Результат 15
Columns("A").ColumnWidth = 10
MsgBox Columns("A").Width      ' Результат 58.2
```

Ширина столбца A, полученная методом ColumnWidth, равна 9,6, в то время как метод Width дает в результате 15. В рассматриваемом случае стилем Normal был назван шрифт Arial размером 10 точек. Однако если свойство ColumnWidth возвращает ширину столбца в единицах, а свойство Width — в точках, дублирование одного из этих свойств должно приводить к дублированию второго.

К счастью, для выяснения истинного положения дел достаточно школьного курса алгебры. Создается впечатление, что свойство Width учитывает также пробелы, расположенные справа и слева от группы символов. Чтобы подтвердить эту догадку, выполним небольшие вычисления, которые вы вполне можете пропустить, если они вам не интересны.

Предположим, что свойство Width не только подсчитывает суммарную ширину группы символов, но и учитывает дополнительные p точек пробелов, расположенных справа и слева от этой группы. Соответственно, формула для вычисления ширины будет выглядеть следующим образом:

$$\text{Width} = 2 \times p + \text{ColumnWidth} \times w$$

Здесь параметр w является истинным значением ширины символа 0. Возьмем значения из примера кода и получим:

$$9,6 = 2 \times p + 1 \times w$$

$$15 = 2 \times p + 2 \times w$$

Вычитание первого уравнения из второго дает:

$$5,4 = w$$

Подставив это значение в первое уравнение, найдем величину параметра p . Она равна 2,1.

Соответственно, формула для стиля Normal, представляющего собой шрифт Arial величиной 10 точек:

$$\text{Width} = 4,2 + \text{ColumnWidth} \times 5,4$$

Таким образом, для свойства ColumnWidth получим:

$$\text{Width} = 4,2 + 10 \times 5,4 = 58,2$$

как и в примере кода!

То есть мы проверили (но не доказали) предположение, что свойство `Width` не только измеряет ширину каждого из символов, но и учитывает пробелы, расположенные справа и слева. В данном случае величина пробелов была равна 2,1.

Метод Consolidate

Метод `Consolidate` объединяет (или комбинирует) данные из нескольких диапазонов, которые могут быть даже расположены на различных листах. Вот его синтаксис:

```
RangeObject.Consolidate(Sources, Function, _  
    TopRow, LeftColumn, CreateLinks)
```

Параметр *Sources* задает источник комбинируемых диапазонов. Это должен быть массив ссылок в стиле R1C1. Причем в ссылки требуется включить полный маршрут доступа к объединяемым диапазонам.

Параметр *Function* указывает на способ объединения данных. Он может принимать значения констант из следующего перечисления:

```
Enum xlConsolidationFunction  
    xlAverage = -4106  
    xlCount = -4112  
    xlCountNums = -4113  
    xlMax = -4136  
    xlMin = -4139  
    xlProduct = -4149  
    xlStDev = -4155  
    xlStDevP = -4156  
    xlSum = -4157  
    xlUnknown = 1000  
    xlVar = -4164  
    xlVarP = -4165  
End Enum
```

По умолчанию этот параметр имеет значение `xlAverage`.

Если требуется, чтобы объединение происходило на основе заголовков столбцов в верхней строке комбинированного диапазона, параметр *TopRow* должен иметь значение `True`. По умолчанию этот параметр имеет значение `False`, и объединение данных происходит согласно каким-либо относительным признакам. Другими словами, когда это свойство имеет значение `True`, приложение комбинирует столбцы с одинаковыми заголовками, даже если они располагаются в разных местах.

Если требуется объединить данные на основе заголовков строк, расположенных в левом столбце диапазона объединения, параметр *LeftColumn* должен иметь значение `True`. По умолчанию этот параметр имеет значение `False`, и объединение данных происходит согласно их относительным признакам.

Чтобы использовать при объединении ссылки на листы, параметру *CreateLinks* следует присвоить значение `True`. По умолчанию этот параметр имеет значение `False`, и при объединении происходит копирование данных.

Для иллюстрации рассмотрим листы, показанные на рис. 19.7 и 19.8. Обратите внимание на порядок следования столбцов.

Код:

```
Worksheets("Sheet1").Range("A1").Consolidate _  
    Sources:=Array("Sheet2!R1C1:R3C3", _  
    "Sheet3!R1C1:R3C3"), Function:=xlSum
```

даст результат, показанный на рис. 19.9. Причем данные окажутся на листе `Sheet1`.

	A	B	C
1	John	Mary	Henry
2	1	2	3
3	4	5	6
4			

Рис. 19.7. Лист Sheet2 до объединения

	A	B	C
1	John	Henry	Mary
2	10	20	30
3	40	50	60
4			

Рис. 19.8. Лист Sheet3 до объединения

	A	B	C
1			
2	11	22	33
3	44	55	66
4			

Рис. 19.9. Вид листа Sheet1 после объединения. Свойство TopRow имеет значение False

Присвоение свойству TopRow значения True:

```
Worksheets("Sheet1").Range("A1").Consolidate
Sources:=Array("Sheet2!R1C1:R3C3",
"Sheet3!R1C1:R3C3"), Function:=xlSum,
TopRow:=True.
```

приведет к результату, показанному на рис. 19.10, так как данные были скомбинированы на основе заголовков, указанных в первой строке.

	A	B	C
1	John	Mary	Henry
2	11	32	23
3	44	65	56
4			

Рис. 19.10. Вид листа Sheet1 после присвоения свойству TopRow значения True

Методы Copy и Cut

Синтаксис метода Copy в применении к объекту Range выглядит следующим образом:

```
RangeObject.Copy(Destination).
```


где параметр *Destination* является объектом *Range*, определяющим диапазон, в который будет осуществляться копирование. При пропуске этого аргумента приложение скопирует данные в буфер. Например, код:

```
Range("A1:C3").Copy Range("D5")
```

копирует диапазон A1:C3 в диапазон аналогичного размера, верхний левый угол которого находится в ячейке D5. То же самое правило применимо и в случае, когда копирование осуществляется с помощью пользовательского интерфейса. В частности, если диапазон назначения содержит более одной ячейки, он должен иметь тот же размер, что и диапазон-источник, иначе появится сообщение об ошибке.

Метод *Cut* имеет сходный синтаксис:

```
RangeObject.Cut(Destination).
```

но не копирует диапазон, а вырезает его.

Метод CopyFromRecordset

Пользователям, знакомым с моделью объектов DAO, не надо объяснять, что метод *CopyFromRecordset* позволяет копировать содержимое объекта DAO *Recordset* на лист, начиная с верхнего левого угла заданного диапазона. Однако если объект *Recordset* содержит поля с OLE-объектами, данный метод работать не будет.

Для иллюстрации рассмотрим код, который требует, чтобы ссылка на модель Microsoft DAO задавалась в диалоговом окне *References* меню *Tools* редактора VBA:

```
Dim rs As Recordset
Set rs = _
    DBEngine.OpenDatabase("d:\excel\excel.mdb"). _
    OpenRecordset("Objects")
Range("A1").CopyFromRecordset(rs, 10, 10)
```

Этот код открывает базу данных Access с именем d:\excel\excel.mdb, создает набор записей на основе таблицы *Objects*, а затем копирует первые 10 столбцов первых 10 строк набора записей в активный лист, начиная с ячейки A1.

В общем случае копирование начинается с активной строки объекта *Recordset*. В нашем примере это первая строка, так как набор записей открыт впервые.

Метод CreateNames

Метод *CreateNames* создает имена на основе текстовых подписей в указанных пользователем ячейках. Вот его синтаксис:

```
RangeObject.CreateNames(Top, Left, Bottom, Right)
```

Все параметры являются необязательными и по умолчанию имеют значение *False*. Если один из них принимает значение *True*, тогда соответствующая строка (*Top* или *Bottom*) или соответствующий столбец (*Left* или *Right*) используются в качестве источника имен для именованного диапазона. Если же все параметры имеют значение *False*, приложение пытается угадать место расположения источника имен. Но я настоятельно не рекомендую полагаться на это обстоятельство.

Для иллюстрации рассмотрим следующий код:

```
Range("A1:C3").CreateNames Top:=True,
```

который в применении к листу, показанному на рис. 19.8, определит три именованных диапазона. Например, диапазон A2:A3 получит имя John.

Свойство CurrentRegion

Крайне полезное свойство CurrentRegion возвращает объект Range, который представляет собой рассматриваемую область, то есть область, окруженную ближайшими пустыми строками и столбцами. Для иллюстрации рассмотрим следующий код:

```
ActiveCell.CurrentRegion.Select
```

Если применить его к листу, показанному на рис. 19.11, выделенной окажется прямоугольная область A2:C4.

	A	B	C	D
1				
2	1		2	
3		0		
4	3		4	
5				
6	5	6	7	
7				

Рис. 19.11. Иллюстрация работы метода CurrentRegion

Метод Delete

Метод Delete удаляет ячейки диапазона. Вот его синтаксис:

```
RangeObject.Delete(Shift)
```

Необязательный параметр *Shift* задает способ смещения ячеек, замещающих удаленные. Он может принимать значения констант из следующего перечисления:

```
Enum xlDeleteShiftDirection
    xlShiftUp = -4162
    xlShiftToLeft = -4159
End Enum
```

Если этот аргумент пропущен, приложение сместит ячейки, исходя из формы удаленной области. Так как это дает совершенно непредсказуемый результат, не рекомендуется пропускать данный аргумент!

Свойства Dependents и DirectDependents

Свойство Dependents возвращает объект Range, представляющий все зависимые ячейки. Для иллюстрации рассмотрим рис. 19.12 с формулами, находящимися в каждой из ячеек.

Следующий код выделяет ячейки B1, B2, C4 и C6:

```
Range("A1").Dependents.Select
```

Обратите внимание на то, что ячейка C1 не зависит от ячейки A1 напрямую.

А вот пример кода, выделяющего только ячейки, непосредственно зависящие от ячейки A1, то есть ячейки B1, B2 и C4:

```
Range("A1").DirectDependents.Select
```


	A	B	C
1	1	=A1	
2		=A1*2	
3			
4			=A1*3
5			
6			=B1*2

Рис. 19.12. Иллюстрация свойства Dependents

Свойства Precedents и DirectPrecedents

Свойства Precedents и DirectPrecedents производят действие, обратное действию свойств Dependents и DirectDependents. Например, код:

```
Range("C6").Precedents.Select
```

выделяет ячейки B1 и A1, в то время как код:

```
Range("C6").DirectPrecedents.Select
```

выделяет ячейку B1.

Свойство End

Свойство End возвращает объект Range, представляющий собой ячейки на конце некоего диапазона, имитируя клавиатурные комбинации. Вот его синтаксис:

```
RangeObject.End(Direction).
```

где объект RangeObject является ссылкой на одну из ячеек, а параметр *Direction* принимает значения констант из следующего перечисления:

```
Enum XlDirection
    xlUp = -4162          ' Ctrl-^
    xlToRight = -4161     ' Ctrl->
    xlToLeft = -4159      ' Ctrl-<
    xlDown = -4121        ' Ctrl-v
End Enum
```

Соответственно, к примеру, код:

```
Range("C4").End(xlToRight).Select
```

выделяет крайнюю справа ячейку в четвертом ряду, которая является последней непустой, начиная с ячейки C4. То есть это ячейка, расположенная непосредственно слева от первой пустой ячейки в четвертом ряду и следующая за ячейкой C4.

Свойства EntireColumn и EntireRow

Свойство EntireColumn возвращает объект Range, состоящий из столбца или столбцов, которые содержат указанный пользователем диапазон. Соответственно, свойство EntireRow возвращает объект Range, состоящий из строки или строк, содержащих указанный пользователем диапазон.

Например, код:

```
Range("A1:A3").EntireRow.Select
```

приводит к выделению первых трех рядов текущего листа.

Методы Fill

В объектной модели Excel существуют четыре варианта метода Fill: FillDown, FillUp, FillRight и FillLeft. Они действуют практически одинаково, поэтому достаточно описать метод FillDown.

Метод FillDown заполняет указанный пользователем диапазон, начиная с верхней ячейки. Значения, формулы и форматирование ячейки или ячеек в верхнем ряду диапазона копируются в остальные ряды. Вот синтаксис этого метода:

```
RangeObject.FillDown
```

Например, код:

```
Range("B3:D5").FillDown
```

дублирует значение из ячеек B3:B5 в ячейки C3:C5 и D3:D5.

Метод Find

Метод Find возвращает первую ячейку рассматриваемого диапазона, которая удовлетворяет заданному критерию. При отсутствии таких ячеек метод возвращает значение Nothing. В любом случае это не влияет на выделенный набор или активную ячейку.

Вот синтаксис метода Find:

```
RangeObject.Find(What, After, LookIn, LookAt, _  
SearchOrder, SearchDirection, MatchCase, MatchByte)
```

Все параметры, кроме параметра *What*, являются необязательными. Этот параметр задает критерий поиска и может представлять собой строку или любой другой действительный тип данных Excel.

Параметр *After* указывает на ячейку, после которой должен начинаться поиск. Если поиск осуществляется с помощью пользовательского интерфейса, он начинается с активной ячейки. Если этот аргумент пропущен, поиск начинается с ячейки, расположенной в верхнем левом углу заданного пользователем диапазона.

Параметр *LookIn* может принимать значения констант из следующего перечисления:

```
Enum XlFindLookIn  
xlValues = -4163  
xlComments = -4144  
xlFormulas = -4123  
End Enum
```

Соответственно, поиск осуществляется среди значений, формул и комментариев.

Параметр *LookAt* определяет, должно ли содержимое ячейки полностью совпадать со значением параметра *What* или же допустимо частичное совпадение. Он принимает значения констант из следующего перечисления:

```
Enum XlLookAt  
xlWhole = 1  
xlPart = 2  
End Enum
```

Параметр *SearchOrder* принимает значения констант из следующего перечисления:

```
Enum XlSearchOrder  
xlByRows = 1
```



```
xlByColumns = 2  
End Enum
```

Значение *xlByRows* задает поиск по строкам, а значение *xlByColumns* — по столбцам.

Параметр *SearchDirection* принимает значения констант из следующего перечисления:

```
Enum xlSearchDirection  
    xlNext = 1           ' Значение по умолчанию  
    xlPrevious = 2  
End Enum
```

Соответственно, по умолчанию переход осуществляется к следующей ячейке, но можно присвоить параметру *SearchDirection* значение *xlPrevious*, в результате чего переход будет осуществляться к предыдущей ячейке.

Для организации поиска с учетом регистра букв параметру *MatchCase* требуется присвоить значение *True*. Параметр *MatchByte* используется только в версии Microsoft Excel Far East. Подробности можно узнать в справочной документации.

Метод *Find* имеет следующие особенности:

- Значения параметров *LookIn*, *LookAt*, *SearchOrder*, *MatchCase* и *MatchByte* сохраняются при каждом вызове метода *Find* и повторно используются при следующем вызове метода. Сюда не входит параметр *SearchDirection*. Кроме того, настройки этих аргументов меняют вид соответствующих настроек в диалоговом окне *Find* (Найти). Обратное также верно. Значит, эти параметры стоит каждый раз указывать заново, потому что пользователь вполне может изменить их в диалоговом окне *Find* (Найти).
- Методы *FindNext* и *FindPrevious*, описанные в следующем разделе, можно использовать для повторного поиска.
- Когда в процессе поиска достигается конец указанного диапазона, происходит возврат к началу последнего. Если подобной функции не требуется, необходимо использовать различные диапазоны.
- Чтобы найти ячейки, соответствующие более сложному критерию, например, с использованием групповых символов, требуется задействовать цикл *For Each* и оператор *Like*. Например, следующий код ищет все ячейки в диапазоне *A1:C5*, содержимое которых начинается с буквы *A*. Затем начертание шрифта в этих ячейках меняется на полужирное:

```
Dim c As Range  
For Each c In [A1:C5]  
    If c.Value Like "A*" Then  
        c.Font.Bold = True  
    End If  
Next
```

Методы *FindNext* и *FindPrevious*

Метод *FindNext* продолжает поиск, начатый методом *Find*, возвращая следующую ячейку, соответствующую указанному критерию. Вот его синтаксис:

RangeObject.FindNext(After)

Если параметр *After* не указан, поиск будет начат с верхнего левого угла заданного диапазона. Соответственно, метод *FindNext* отличается от метода *Find* только

тем, что в нем применяются все параметры (кроме *After*), определенные при предыдущем использовании метода *Find*.

Чтобы продолжить поиск с последней найденной ячейки, требуется использовать эту ячейку в качестве аргумента *After*. Например, следующий код ищет в верхней строке все ячейки, содержащие значение 0, и удаляет это значение:

```
Dim c As Range
Dim sFirstHit As String ' Адрес первой совпавшей ячейки
With Rows(1)
    Set c = .Find(0, LookIn:=xlValues)
    If Not c Is Nothing Then
        sFirstHit = c.Address
        Do
            ' Изменение содержимого ячейки
            c.Value = ""
            ' Поиск следующей ячейки
            Set c = .FindNext(c)
        Loop While Not c Is Nothing
    End If
End With
```

Метод *FindPrevious* имеет следующий синтаксис:

```
RangeObject.FindPrevious(Before)
```

и работает совершенно аналогично методу *FindNext*, просто поиск осуществляется в обратном направлении и начинается с ячейки, заданной параметром *Before*.

Свойства *Formula* и *FormulaR1C1*

Свойство *Formula* возвращает или задает формулу или значение для каждой ячейки указанного пользователем диапазона. Формула должна быть записана в стиле A1 с предшествующим знаком равенства.

Например, строка:

```
Range("A1").Formula = "=Sum(A2:A3)"
```

задает формулу в ячейке A1. А строка:

```
Range("A1:C1").Formula = "=Sum(A2:A3)"
```

помещает формулу в ячейки A1:C1, но так как в формуле содержатся относительные ссылки, они изменяются обычным образом. Если же требуется поместить в каждую ячейку неизменяемую формулу, следует использовать массив. Например:

```
Range("A1:C1").Formula =  
Array("=Sum(A2:A3)", "=Sum(A2:A3)", "=Sum(A2:A3)")
```

Для возврата массива также можно воспользоваться свойством *Formula*. Для иллюстрации рассмотрим лист, показанный на рис. 19.13. Код:

```
Dim a As Variant  
a = Range("A1:C2").Formula
```

указывает, что переменная *a*, принадлежащая к типу *Variant*, является массивом, соответственно, выражение *a(2,3)* дает в результате значение 7.

Если в ячейке содержится константа, свойство *Formula* возвращает эту константу. Назначить константу ячейке можно, написав:

```
Range("A1").Formula = 1
```


В случае с пустой ячейкой свойство Formula возвращает пустую строку. Когда ячейка содержит формулу, возвращается формула в виде строки в том виде, в каком она отображается в строке формул, включая знак равенства.

	A	B	C
1	2	3	4
2	5	6	7
3			

Рис. 19.13. Иллюстрация работы со свойством Formula

Если задать свойство Formula (или свойство Value) в виде даты, приложение проверит, отформатирована ли ячейка в соответствии с одним из форматов даты или времени. В случае отсутствия форматирования Excel использует заданный по умолчанию короткий формат даты.

Свойство FormulaR1C1 является аналогом свойства Formula, но принимает и возвращает формулы в стиле R1C1.

Свойство FormulaArray

Свойство FormulaArray возвращает или определяет формулу массива для диапазона в стиле R1C1. Для иллюстрации рассмотрим лист, показанный на рис. 19.14. Код: Range("A9:C11").FormulaArray = "=A1:C3 + A5:C7"

привел к появлению значений в ячейках A9:C11 на рис. 19.14. Формула слева указывает, что нужно сложить содержимое соответствующих ячеек верхнего массива 3 × 3 с содержимым ячеек расположенного в центре массива 3 × 3, помещая результат в нижний массив 3 × 3.

	A	B	C
1	1	2	3
2	4	5	6
3	7	8	9
4			
5	10	11	12
6	13	14	15
7	16	17	18
8			
9	11	13	15
10	17	19	21
11	23	25	27

Рис. 19.14. Иллюстрация применения свойства FormulaArray

Учтите, что код:
Debug.Print Range("A9").FormulaArray
выводит на печать формулу массива:
=A1:C3 + A5:C7

Свойство FormulaHidden (чтение/запись, тип Boolean)

Свойство FormulaHidden возвращает или задает скрытое состояние формул в ячейке, к которой оно применяется. Это эквивалентно установке флажка Hidden (Скрыть формулы) на вкладке Protection (Защита) диалогового окна Format cells (Формат ячеек).

Следует различать свойства FormulaHidden и Hidden. Последнее применяется к областям, состоящим из целых строк (или целых столбцов), и определяет, являются ли эти строки (или столбцы) видимыми.

Свойство HasFormula (только чтение)

Свойство HasFormula возвращает значение True, если все ячейки рассматриваемого диапазона содержат формулы. Если формул не содержится ни в одной из ячеек диапазона, возвращается значение False. Во всех остальных случаях возвращается значение Null.

Свойство HorizontalAlignment

Свойство HorizontalAlignment возвращает или задает горизонтальное выравнивание во всех ячейках указанного пользователем диапазона. Оно может принимать значения констант из следующего перечисления:

```
Enum xlHAlign
    xlHAlignRight = -4152
    xlHAlignLeft = -4131
    xlHAlignJustify = -4130
    xlHAlignDistributed = -4117      'Для Far East Excel
    xlHAlignCenter = -4108
    xlHAlignGeneral = 1
    xlHAlignFill = 5
    xlHAlignCenterAcrossSelection = 7
End Enum
```

Обратите особое внимание на константу xlHAlignCenterAcrossSelection, которая используется при выравнивании заголовков, растянутых на несколько ячеек.

Свойство IndentLevel и метод InsertIndent

Свойство IndentLevel возвращает или задает левый отступ в каждой ячейке рассматриваемого диапазона и может принимать целочисленные значения от 0 до 15. Присвоение других значений приводит к сообщению об ошибке. Отступы в основном используются при выравнивании содержимого ячеек или при форматировании текста.

Чтобы сделать отступ в ячейке A1 равным 10, требуется написать:

```
Range("A1").IndentLevel = 10
```

К сожалению, в документации не указано абсолютное значение единичного отступа, но можно воспользоваться относительными величинами. Очевидно, что отступ второго уровня в два раза больше отступа первого уровня.

Альтернативой свойству IndentLevel является метод InsertIndent. Вот его синтаксис:

```
RangeObject.InsertIndent(InsertAmount),
```


где аргумент *InsertAmount* является целым числом от 0 до 15. Однако в данном случае этот параметр указывает величину изменения текущего отступа для рассматриваемого диапазона.

Метод Insert

Метод Insert включает в лист ячейку или диапазон, смещая существующие ячейки. Вот его синтаксис:

```
RangeObject.Insert(Shift).
```

где параметр *Shift* может принимать значения констант из следующего перечисления:

```
Enum XlInsertShiftDirection  
    xlShiftToRight = -4161  
    xlShiftDown = -4121  
End Enum
```

В случае пропуска этого аргумента Excel выбирает направление смещения, исходя из формы вставляемого диапазона. Как всегда, не следует полагаться на выбор программы.

Свойство Locked

Свойство Locked возвращает состояние Locked ячеек рассматриваемого диапазона и может использоваться для блокировки последних. Если диапазон одновременно содержит заблокированные и свободные ячейки, это свойство возвращает значение Null.

Методы и свойства, связанные со слиянием

Распространенной практикой является создание слитых ячеек, то есть получение одной ячейки путем комбинации их набора. Это требуется, например, при написании заголовков.

Метод Merge создает слитую ячейку из указанного пользователем диапазона. Вот его синтаксис:

```
RangeObject.Merge(Across).
```

где необязательный параметр *Across* принадлежит к типу Variant. Если требуется слить ячейки в каждом ряду, оставив разделение на ряды, этому параметру нужно присвоить значение True. По умолчанию он имеет значение False. При этом происходит слияние всего диапазона в единую ячейку. Имейте в виду, что слитая ячейка будет содержать данные только из левой верхней ячейки исходного диапазона. Данные из остальных ячеек будут потеряны.

Метод UnMerge разделяет слитую область на отдельные ячейки. Вот его синтаксис:

```
RangeObject.UnMerge
```

Если объект RangeObject содержит ячейки, расположенные в слитом диапазоне, даже если при этом в него входят не все ячейки слитой области или, наоборот, входят свободные ячейки, метод UnMerge все равно разделит слитую область на отдельные ячейки. В процессе применения этого метода к диапазону, не содержащему слитых ячеек, сообщения об ошибке не появляется. Просто метод не производит никакого эффекта.

Свойство `MergeArea` применимо только к диапазонам, состоящим из одной ячейки. В противном случае появляется сообщение об ошибке. Это свойство возвращает объект `Range`, представляющий собой слитый диапазон, который содержит данную ячейку. Если ячейка не является частью слитого диапазона, свойство `MergeArea` возвращает ее саму.

Свойство `MergeCells` возвращает значение `True`, если указанный пользователем диапазон содержится внутри слитого диапазона ячеек. Когда этот диапазон содержит не только слитые, но и свободные ячейки, свойство возвращает значение `Null`.

Свойства `Next` и `Previous`

В применении к объекту `Range` свойство `Next` возвращает ячейку, которая становится активной после нажатия клавиши `Tab`, хотя при этом фактического выделения данной ячейки не происходит. Соответственно, на незащищенном листе это свойство возвращает ячейку, расположенную непосредственно справа от левой верхней ячейки указанного пользователем диапазона. На защищенном листе оно возвращает следующую незаблокированную ячейку.

Аналогично свойство `Previous` эмулирует нажатие комбинации клавиш `Shift + Tab`, возвращая, но не выделяя соответствующую ячейку.

Свойство `NumberFormat`

Свойство `NumberFormat` возвращает или задает строку с форматированием числовых данных в указанном пользователем диапазоне. Если не все ячейки этого диапазона отформатированы одинаковым образом, свойство возвращает значение `Null`.

Проще всего определить нужную строку с форматированием, записав соответствующий макрос и воспользовавшись диалоговым окном `Format` (Формат). Именно в коде макроса вы найдете нужную строку.

Метод `Parse`

Метод `Parse` разбирает данные по столбцам и распределяет содержимое диапазона, заполняя соседние столбцы. Вот его синтаксис:

```
RangeObject.Parse(ParseLine, Destination)
```

Объект `RangeObject` не может иметь ширину более одного столбца.

Параметр `ParseLine` является строкой, содержащей квадратные скобки, которые указывают на место разделения данных в ячейках. Например, строка:

```
[xxx] [xxx]
```

приводит к тому, что метод `Parse` включает первые три символа из каждой ячейки в первый столбец назначенного диапазона, пропускает четвертый символ, а затем включает следующие три символа во второй столбец. Любые дополнительные символы, расположенные за шестью первыми, не будут включены в диапазон. Это делает метод `Parse` крайне полезным при разборе данных с фиксированной длиной. В итоге каждая ячейка получает данные одной и той же длины.

Параметр `Destination` является объектом `Range`, который представляет собой верхний левый угол диапазона назначения для разделяемых данных. Если этот аргумент пропущен, Excel использует столбец исходных данных в качестве первого столбца диапазона назначения.

Метод PasteSpecial

Метод PasteSpecial вставляет данные из буфера обмена в указанный пользователем диапазон. Вот его синтаксис:

```
RangeObject.PasteSpecial(Paste, Operation, SkipBlanks, Transpose)
```

Параметр *Paste* указывает тип включаемых данных и может принимать значения констант из следующего перечисления:

```
Enum XlPasteType
    xlPasteValues = -4163
    xlPasteComments = -4144
    xlPasteFormulas = -4123
    xlPasteFormats = -4122
    xlPasteAll = -4104 ' Значение по умолчанию
    xlPasteAllExceptBorders = 6
End Enum
```

Необязательный параметр *Operation* определяет операцию вставки и может принимать значения констант из следующего перечисления:

```
Enum XlPasteSpecialOperation
    xlPasteSpecialOperationNone = -4142 ' Значение по умолчанию
    xlPasteSpecialOperationAdd = 2
    xlPasteSpecialOperationSubtract = 3
    xlPasteSpecialOperationMultiply = 4
    xlPasteSpecialOperationDivide = 5
End Enum
```

Чтобы из буфера обмена не вставлялись пустые ячейки, параметру *SkipBlank* следует присвоить значение True. Предположим, что ячейка, содержимое которой требуется включить из буфера обмена в ячейку D5, является пустой. Если параметр *SkipBlank* имеет значение False, содержимое ячейки D5 будет заменено пустой ячейкой. Соответственно, чтобы сохранить содержимое ячейки без изменений, параметру *SkipBlank* нужно присвоить значение True.

Необязательному параметру *Transpose* присваивается значение True, если при вставке диапазона требуется поменять местами строки и столбцы. По умолчанию этот параметр имеет значение False.

Метод PrintOut

Метод PrintOut выводит на печать диапазон. Впрочем, он применим и к набору других объектов, включая объекты Workbook, Worksheet и Chart. Вот его синтаксис:

```
RangeObject.PrintOut(From, To, Copies, Preview, _
    ActivePrinter, PrintToFile, Collate)
```

Все параметры этого метода являются необязательными.

Параметр *From* указывает номер первой распечатываемой страницы, а параметр *To*, соответственно, — номер последней. Пропуск этих параметров приводит к распечатке всего объекта (диапазона, листа и т. п.).

Параметр *Copies* определяет число распечатываемых копий. По умолчанию он имеет значение 1.

Присвоение параметру *SetPreview* значения True дает возможность осуществлять предварительный просмотр распечатываемого материала. По умолчанию этот параметр имеет значение False.

Параметр *ActivePrinter* задает имя используемого принтера.

Присвоение параметру *PrintToFile* значения *True* приводит к тому, что Excel выводит на печать файл. При этом пользователь должен указать имя выходного файла. К сожалению, не существует способа задать это имя внутри кода.

Параметр *Collate* должен иметь значение *True*, чтобы осуществить сортировку множества многостраничных копий.

Метод PrintPreview

Метод *PrintPreview* дает возможность использовать функцию предварительного просмотра указанного пользователем диапазона. Этот метод также применим к различным объектам, как и метод *PrintOut*. Вот его синтаксис:

```
RangeObject.PrintPreview
```

Метод Replace

Метод *Replace* используется для поиска и замены указанных пользователем данных во всех ячейках диапазона. Он не оказывает влияния на выделенный набор ячеек или на активную ячейку. Вот его синтаксис:

```
RangeObject.Replace(What, Replacement, LookAt, _  
SearchOrder, MatchCase, MatchByte)
```

Параметр *What* определяет данные, которые требуется найти, а параметр *Replacement* — данные для замены. Причем тип данных может быть произвольным.

Параметр *LookAt* определяет, должно ли содержимое ячейки полностью совпадать со значением параметра *What* или же допустимо частичное совпадение. Он принимает значения констант из следующего перечисления:

```
Enum XlLookAt  
xlWhole = 1  
xlPart = 2  
End Enum
```

Параметр *SearchOrder* принимает значения констант из следующего перечисления:

```
Enum XlSearchOrder  
xlByRows = 1  
xlByColumns = 2  
End Enum
```

Значение *xlByRows* задает поиск по строкам, а значение *xlByColumns* — по столбцам.

Значения параметров *LookIn*, *LookAt*, *SearchOrder*, *MatchCase* и *MatchByte* сохраняются при каждом вызове метода *Replace* и повторно используются при его следующем вызове. Кроме того, настройки этих аргументов меняют вид соответствующих настроек в диалоговом окне *Find and Replace* (Найти и заменить). Обратное также верно. Соответственно, эти параметры стоит каждый раз указывать заново, потому что пользователь вполне может изменить их в диалоговом окне *Find and Replace* (Найти и заменить).

Если содержимое аргумента *What* находится хотя бы один раз, метод *Replace* возвращает значение *True*.

Метод Select

Метод `Select` используется для выделения заданного пользователем диапазона. Впрочем, он применяется к 81 различному объекту Excel. Для объекта `Range` его синтаксис выглядит следующим образом:

```
RangeObject.Select
```

Учтите, что этот метод позволяет выделять набор ячеек, в то время как метод `Activate` активизирует только одну ячейку.

Свойство ShrinkToFit

Свойству `ShrinkToFit` присваивается значение `True`, когда требуется уменьшить размер шрифта всего текста в диапазоне до такой степени, чтобы его ширина совпала с шириной столбца. Соответственно, если это свойство присуще всем ячейкам диапазона, оно возвращает значение `True`. Когда этим свойством не обладает ни одна ячейка диапазона, оно возвращает значение `False`. В остальных случаях возвращается значение `Null`.

Метод Sort

Метод `Sort` используется для сортировки диапазона. Вот его синтаксис:

```
RangeObject.Sort(Key1, Order1, Key2, Type, Order2, Key3, Order3, _  
    Header, OrderCustom, MatchCase, Orientation, SortMethod, _  
    IgnoreControlCharacters, IgnoreDiacritics, IgnoreKashida)
```

Для сортировки можно использовать три ключа, задаваемые параметрами `Key1`, `Key2` и `Key3`, которым допустимо присвоение как текста, так и объекта `Range`. Порядок сортировки определяется параметром `Order`, который может принимать значения констант из следующего перечисления:

```
Enum XlSortOrder  
    xlAscending = 1          ' Значение по умолчанию  
    xlDescending = 2  
End Enum
```

В первом случае сортировка осуществляется по возрастанию, а во втором — по убыванию.

Необязательный параметр `Type` используется только при сортировке сводных таблиц.

Необязательный параметр `Header` может принимать значения констант из следующего перечисления:

```
Enum XlYesNoGuess  
    xlGuess = 0  
    xlYes = 1  
    xlNo = 2                ' Значение по умолчанию  
End Enum
```

Он указывает на то, является ли первая строка заголовком и участвует ли она в сортировке. Если этот параметр имеет значение `xlGuess`, программа Excel сама определяет, является ли первая строка заголовком.

Необязательный параметр `OrderCustom` является целым числом, представляющим собой нестандартный порядок сортировки. Однако в документации Microsoft

не существует дальнейших уточнений по поводу данного параметра, поэтому наиболее разумным кажется его пропуск.

Если значение необязательного параметра *MatchCase* равно True, сортировка происходит с учетом регистра букв. Предположим, что ячейка A1 содержит текст «ААА», а ячейка A2 — текст «aaa». Код:

```
Range("A1:A2").Sort Key1:=Cells(1, 1), MatchCase:=True
```

просто поменяет содержимое этих двух ячеек, зато код:

```
Range("A1:A2").Sort Key1:=Cells(1, 1), MatchCase:=False
```

не окажет никакого эффекта.

Необязательный параметр *Orientation* определяет, каким образом осуществляется сортировка: по строкам или по столбцам. Он может принимать значения констант из следующего перечисления:

```
Enum XlSortOrientation
    xlSortColumns = 1
    xlSortRows = 2
End Enum
```

Например, строка:

```
Range("A1:B2").Sort Key1:=Rows(1), Orientation:=xlSortColumns
```

сортирует столбцы в диапазоне A1:B2. При этом в качестве ключа для сортировки используется первая строка.

Метод SpecialCells

Метод *SelectSpecial* возвращает объект *Range*, представляющий собой все ячейки, содержимое которых принадлежит к заданному типу и имеет заданное значение. Вот его синтаксис:

```
RangeObject.SpecialCells(Type, Value)
```

Параметр *Type* задает тип ячеек, которые входят в объект *RangeObject*. Он может принимать значения констант из следующего перечисления:

```
Enum XlCellType
    xlCellTypeComments = -4144 'Ячейки с комментариями
    xlCellTypeFormulas = -4123 'Ячейки с формулами
    xlCellTypeConstants = 2 'Ячейки с константами
    xlCellTypeBlanks = 4 'Пустые ячейки
    xlCellTypeLastCell = 11 'Последняя ячейка диапазона
    xlCellTypeVisible = 12 'Все видимые ячейки
End Enum
```

Например, код:

```
Range("A1:D10").SpecialCells(xlCellTypeBlanks).Select
```

выделяет все пустые ячейки в диапазоне A1:D10.

Необязательный параметр *Value* применяется в тех случаях, когда параметр *Type* имеет значение *xlCellTypeConstants* или *xlCellTypeFormulas* и более точно указывает на тип возвращаемой ячейки. Он может принимать значения одной или нескольких констант из следующего перечисления:

```
Enum XlSpecialCellsValue
    xlNumbers = 1 'Числа
    xlTextValues = 2 'Текстовые значения
    xlLogical = 4 'Логические значения
```



```
xlErrors = 16      'Ошибки
End Enum
```

Например, код:

```
Range("A1:D10").SpecialCells(xlCellTypeConstants, xlTextValues).Select
```

выделяет только ячейки с текстом (а не с числами) в диапазоне A1:D10.

Метод TextToColumns

Метод TextToColumns разбирает столбец (или столбцы) ячеек, содержащих текст, на несколько отдельных столбцов. Вот его синтаксис:

```
RangeObject.TextToColumns(Destination, DataType, _
    TextQualifier, ConsecutiveDelimiter, Tab, Semicolon, _
    Comma, Space, Other, OtherChar, FieldInfo)
```

Все параметры этого метода являются необязательными.

Параметр *Destination* представляет собой объект Range, указывающий, куда следует поместить результат разбора. Если объект Range включает несколько ячеек, начальной точкой размещения данных является верхний левый угол диапазона.

Параметр *DataType* задает формат разбираемого на столбцы текста. Он может принимать значения констант из следующего перечисления:

```
Enum xlTextParsingType
    xlDelimited = 1      ' Значение по умолчанию
    xlFixedWidth = 2
End Enum
```

Параметр *TextQualifier* используется для описания текста и принимает значения констант из следующего перечисления:

```
Enum xlTextQualifier
    xlTextQualifierNone = -4142
    xlTextQualifierDoubleQuote = 1      ' Значение по умолчанию
    xlTextQualifierSingleQuote = 2
End Enum
```

Если требуется, чтобы Excel рассматривал последовательные разделители как один, параметру *ConsecutiveDelimiter* следует присвоить значение True. По умолчанию он имеет значение False.

Существует ряд параметров, для которых необходимо, чтобы параметр *DataType* имел значение xlDelimited. Если какой-нибудь из этих параметров имеет значение True, это значит, что Excel должен рассматривать соответствующий символ как разделитель текста. Перечислим эти параметры, имеющие по умолчанию значение False:

- *Tab*. Когда этот параметр принимает значение True, в качестве разделителя используется знак табуляции.
- *Semicolon*. Когда этот параметр принимает значение True, в качестве разделителя используется точка с запятой.
- *Comma*. Когда этот параметр принимает значение True, в качестве разделителя используется запятая.
- *Space*. Когда этот параметр принимает значение True, в качестве разделителя используется пробел.

○ *Other*. Когда данный параметр принимает значение True, в качестве разделителя используется знак, определенный аргументом *OtherChar*. Если этот аргумент содержит несколько знаков, разделителем служит первый из них.

Параметр *FieldInfo* является массивом, который содержит информацию, необходимую для разбора отдельных столбцов данных файла-источника. Интерпретация этого параметра зависит от значения параметра *DataType*.

Если последний параметр имеет значение *xlDelimited*, аргумент *FieldInfo* будет представлять собой массив, размер которого определяется числом столбцов преобразуемых данных. Первый элемент двухэлементного массива представляет собой номер столбца (нумерация начинается с номера 1), а второй — номер, указывающий на способ разбора столбца (табл. 19.3).

Таблица 19.3

Значение	Описание
1	Общий
2	Текст
3	Данные МДГ
4	Данные ДМГ
5	Данные ГМД
6	Данные МГД
7	Данные ДГМ
8	Данные ГДМ
9	Пропустить столбец

Если для рассматриваемого столбца отсутствует двухэлементный массив, разбор столбца производится общим образом. Например, следующее значение параметра *FieldInfo*:

```
Array(Array(1, 2), Array(3, 9))
```

приводит к тому, что первый столбец разбирается как текст, а третий — пропускается. Все остальные столбцы разбираются как общие данные.

Для иллюстрации рассмотрим лист, показанный на рис. 19.15. Код:

```
Range("A1:A3").TextToColumns _  
Destination:=Range("B1"). _  
DataType:=xlDelimited, _  
ConsecutiveDelimiter:=True, Comma:=True, _  
FieldInfo:=Array(Array(1, 2), Array(2, 3))
```

приводит к появлению второго и третьего столбцов. Обратите внимание на то, что ячейки в столбце С отформатированы как даты.

	A	B	C
1	book, 12/4/98	book	12/4/98
2	record, 1/17/98	record	1/17/98
3	car, 11/2/98	car	11/2/98

Рис. 19.15. Лист с текстом, который был разобран по столбцам В и С

С другой стороны, если параметр *DataType* имеет значение *xlFixedWidth*, первый элемент любого двухэлементного массива будет указывать начальное положение символа в столбце (при этом первым символом будет 0), а второй, как и в предыдущем случае, будет определять способ разбора текста.

Для иллюстрации рассмотрим лист, показанный на рис. 19.16. Код:

```
Range("A1:A3").TextToColumns _
Destination:=Range("B1"). _
DataType:=xlFixedWidth, _
FieldInfo:=Array(Array(0, 2), _
Array(1, 9), Array(2, 2), Array(5, 9), _
Array(6, 2))
```

разбирает текст в первом столбце по столбцам В, С и D. Обратите внимание на то, как с помощью массива были удалены дефисы.

	A	B	C	D
1	1-234-567	1	234	567
2	2-435-678	2	435	678
3	5-444-666	5	444	666

Рис. 19.16. Лист с данными фиксированной длины в ячейках A1:A3, которые были разобраны по соседним столбцам

Свойство Value

Свойство *Value* возвращает значение, содержащееся в указанной пользователем ячейке. В случае с пустой ячейкой свойство возвращает пустую строку. Проверить наличие в ячейке содержимого можно таким способом:

```
If Range("A1") = "" Then . . .
```

или

```
If IsEmpty(Range("A1")) Then . . .
```

Если объект *Range* содержит несколько ячеек, свойство *Value* возвращает двумерный массив. Например, возвращаясь к рис. 19.16, код:

```
Dim v As Variant
v = Range("A1:A3").Value
Debug.Print IsArray(v)
Debug.Print v(2, 1) ' Строка 2, столбец 1
```

приведет к выводу на печать следующей информации:

```
True
2-435-678
```

Свойство WrapText

Свойство *WrapText* возвращает или задает значение, которое сообщает приложению Excel о том, следует ли переносить текст в ячейке. Если в некоторых ячейках выделенного диапазона текст переносится, а в некоторых — нет, свойство возвращает значение *Null*. При необходимости Excel изменяет высоту строк диапазона, чтобы уместить перенесенный текст.

Потомки объекта Range

Потомки объекта Range показаны на рис. 19.17.

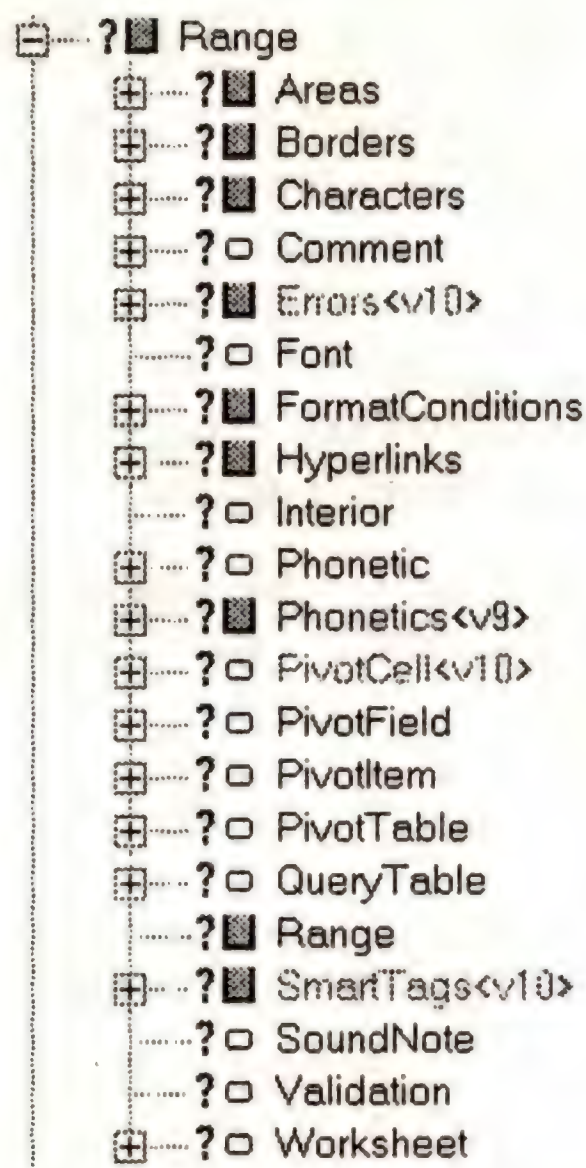


Рис. 19.17. Потомки объекта Range

Соответственно, каждый из этих дочерних объектов является свойством объекта Range, возвращающим объект-потомок. Например, свойство PivotField объекта Range возвращает объект PivotField.

Поговорим подробно о потомках объекта Range.

Семейство Areas

Областью (area) называется непрерывный блок ячеек листа. Но в объектной модели Excel отсутствует объект Area. Вместо него используется объект Range.

При этом любой диапазон состоит из одной или нескольких областей. Семейство объектов Range, которое представляет собой эти области, и составляет семейство Areas для анализируемого диапазона. Для иллюстрации рассмотрим рис. 19.18, полученный в результате работы следующего кода:

```

Dim rng As Range
Set rng = ActiveSheet.Cells.SpecialCells( _
    xlCellTypeConstants, xlNumbers)
rng.Select
  
```

При этом были выделены три области.

Очистить вторую область можно, написав:

```
rng.Areas(2).Clear
```

В результате очистятся ячейки C4 и D4. Область является семейством, нумерация в котором начинается с единицы.

	A	B	C	D
1	1	2		
2	3	4		
3				
4			5	6
5		7		
6		8		
7				

Рис. 19.18. Диапазон, состоящий из трех областей

Впрочем, мне кажется, что обращаться к области по номеру несколько рискованно. В то же время последовательное обращение к областям с помощью цикла For вполне безопасная процедура:

```
Dim rng As Range, r As Range
Set rng = ActiveSheet.Cells.SpecialCells( _
    xlCellTypeConstants.xlNumbers)
For Each r In rng.Areas
    Debug.Print r.Cells.Count
Next
```

Семейство Borders

Каждый диапазон окружен некоей границей. Например, нижняя граница диапазона состоит из нижних границ всех ячеек нижнего ряда этого диапазона. Например, нижняя граница в данном случае:

```
Range("a1:b4, d2:e2")
```

Она показана на рис. 19.19 в виде темных линий.

	A	B	C	D	E
1					
2					
3					
4					
5					

Рис. 19.19. Пример объекта Border

Свойство Border объекта Range возвращает семейство Borders для рассматриваемого диапазона. Это семейство состоит из нескольких объектов, которым соответствуют следующие константы:

```
Enum XlBordersIndex
    xlDiagonalDown = 5
    xlDiagonalUp = 6
    xlEdgeLeft = 7
    xlEdgeTop = 8
    xlEdgeBottom = 9
    xlEdgeRight = 10
    xlInsideVertical = 11
    xlInsideHorizontal = 12
End Enum
```


В документации Excel это перечисление называется `XlBorderType`.

Для иллюстрации рассмотрим код, который присваивает ячейкам диапазона, показанного на рис. 19.19, серый цвет, а нижнюю границу делает красной. Обратите внимание на использование вложенного оператора `With`:

```
With Range("a1:b4, d2:e2")
    .Interior.Color = RGB(196, 196, 196)
    With .Borders(xlEdgeBottom)
        .Weight = xlThick
        .Color = RGB(255, 0, 0)
    End With
End With
```

На рис. 19.20 показан результат присвоения константе `xlEdgeBottom` значения `xlDiagonalDown`. Затем этой константе было присвоено значение `xlInsideVertical`. Полученный результат показан на рис. 19.21.

	A	B	C	D	E
1					
2					
3					
4					
5					

Рис. 19.20. Действие константы `xlDiagonalDown`

	A	B	C	D	E
1					
2					
3					
4					

Рис. 19.21. Действие константы `xlInsideVertical`

Объект Border

В этом разделе мы поговорим о наиболее интересных свойствах и методах объекта `Border`.

Свойство Color

Свойство `Color` определяет или возвращает основной цвет границы. Оно также применимо к семейству `Borders` и задает все вертикальные и горизонтальные линии границ одновременно. Кроме того, это свойство присуще объектам `Font` и `Interior`.

Для иллюстрации рассмотрим код, который производит эффект, показанный на рис. 19.22:

```
With Range("a1:b4, d2:e2")
    .Interior.Color = RGB(196, 196, 196)
    With .Borders
        .Weight = xlThick
        .Color = RGB(255, 0, 0)
    End With
End With
```


	A	B	C	D	E
1					
2					
3					
4					

Рис. 19.22. Назначение свойства Color семейству Borders

Для указания цвета используется функция RGB, имеющая форму:
RGB(*red*, *green*, *blue*),
где аргументы *red*, *green* и *blue* являются целыми числами от 0 до 255 и определяют вклад каждого из компонентов цвета. В табл. 19.3 приведены часто используемые значения цветов.

Таблица 19.3. Часто используемые цвета

Цвет	Red (Красный)	Green (Зеленый)	Blue (Синий)
Черный	0	0	0
Синий	0	0	255
Зеленый	0	255	0
Голубой	0	255	255
Красный	255	0	0
Пурпурный	255	0	255
Желтый	255	255	0
Белый	255	255	255

Серый цвет соответствует равному количеству красного, зеленого и синего компонентов. Например:

```
RGB(196, 196, 196)
```

В данном случае идет речь о 25% серого цвета. Чем больше числа, тем ближе итоговый цвет к белому. К сожалению, Excel округляет все настройки оттенков серого до одного из следующих значений:

- 0% (белый);
- 25%;
- 40%;
- 50%;
- 80%;
- 100% (черный).

В этом можно убедиться, запустив на выполнение следующий код:

```
Dim r As Integer
For r = 1 To 25
    Cells(r, 1).Interior.Color =
        RGB(255 - 10 * r, 255 - 10 * r, 255 - 10 * r)
    Cells(r, 2).Value = 255 - 10 * r
Next
```


Если вы собираетесь часто использовать оттенки серого цвета, добавьте к модулю кода следующие описания констант:

```
Public Const Gray25 = 12632256
Public Const Gray40 = 9868950
Public Const Gray50 = 8421504
Public Const Gray80 = 3355443
```

Свойство ColorIndex

Свойство ColorIndex определяет цвет в цветовой палитре по его индексу. В книге с черно-белыми рисунками нет возможности проиллюстрировать его действие, поэтому рекомендуется воспользоваться документацией Excel, которая сопровождается цветными рисунками. Это свойство может принимать значения констант из следующего перечисления:

```
Enum xlColorIndex
    xlColorIndexNone = -4142      ' Заливка отсутствует
    xlColorIndexAutomatic = -4105 ' Автоматическая заливка
End Enum
```

Свойство LineStyle

Свойство LineStyle возвращает или задает вид границы. Оно может принимать значения констант из следующего перечисления:

```
Enum xlLineStyle
    xlLineStyleNone = -4142      ' Обычная линия
    xlDouble = -4119             ' Двойная линия
    xlDot = -4118                ' Точки
    xlDash = -4115               ' Пунктир
    xlContinuous = 1             ' Непрерывная линия
    xlDashDot = 4                ' Тире, точка
    xlDashDotDot = 5             ' Тире, точка, точка
    xlSlantDashDot = 13          ' Наклонные тире и точка
End Enum
```

Свойство Weight

Свойство Weight возвращает или задает толщину границы. Оно может принимать значения констант из следующего перечисления:

```
Enum xlBorderWeight
    xlMedium = -4138             ' Средняя толщина линии
    xlHairline = 1               ' Самая тонкая из видимых линий
    xlThin = 2                   ' Тонкая линия
    xlThick = 4                  ' Толстая линия
End Enum
```

Объект Characters

Объект Characters представляет собой соприкасающиеся последовательности текстовых символов. Его основным назначением является модификация части текстовой строки. Вот его синтаксис:

```
RangeObject.Characters(start, length).
```

где параметр *Start* указывает на номер первого знака, а параметр *length* — на количество знаков в строке. Для иллюстрации рассмотрим код, выделяющий полужирным шрифтом первое слово в ячейке:


```
Dim rng As Range
Set rng = Range("A1")
rng.Characters(1, InStr(rng.Value, " ") - 1).Font.Bold = True
```

Результат показан на рис. 19.23.

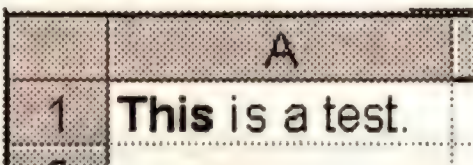


Рис. 19.23. Выделение полужирным шрифтом первого слова в ячейке

Объект Comment

Как вы помните, метод AddComment объекта Range использовался для добавления в диапазон комментариев. Он приводил к появлению объектов Comment, каждый из которых принадлежит к семейству Comments объекта Worksheet.

Для иллюстрации рассмотрим код, создающий комментарий в ячейке A1. Затем указывается текст, который становится видимым примерно за три секунды. Обратите внимание на использование оператора DoEvents, гарантирующего, что комментарий появится до входа в цикл Do. Вы можете попробовать запустить этот код без оператора DoEvents. В моей системе в этом случае текст комментария просто не появлялся. Обратите также внимание на то, что функция Timer возвращает число секунд после полуночи. Соответственно, если трехсекундный интервал придется на полночь, возникнет проблема.

```
Dim tm As Single
tm = Timer
If Range("A1").Comment Is Nothing Then
    Range("A1").AddComment "Комментарий"
End If
Range("A1").Comment.Text "Создан: " & Now
Range("A1").Comment.Visible = True
DoEvents
Do: Loop Until Timer - tm > 3
Range("A1").Comment.Visible = False
```

Объект Font

Свойство Font объекта Range возвращает объект Font, который применяется для контроля вида шрифтов (название шрифта, его размер, цвет и т. п.), используемых в рассматриваемом диапазоне.

Свойства объекта Font перечислены в табл. 19.4.

Таблица 19.4. Свойства объекта Font

Application	FontStyle	Size
Background	Italic	Strikethrough
Bold	Name	Subscript
Color	OutlineFont	Superscript
ColorIndex	Parent	Underline
Creator	Shadow	

Семейство FormatConditions

Приложение Excel позволяет применять к ячейке или к диапазону *условное форматирование*. Под этим термином подразумевается форматирование, которое применяется только в случае соответствия содержимого ячеек определенному условию. Например, можно сделать отрицательные числа красными, положительные — черными, а нули — зелеными.

Свойство FormatConditions объекта Range возвращает семейство FormatConditions, которое может содержать до трех объектов FormatConditions. Каждый из этих объектов представляет собой условие форматирования.

Для добавления объектов FormatConditions в одноименное семейство используется метод Add. Однако попытка добавить более трех объектов приведет к сообщению об ошибке. Вот синтаксис метода Add, применяемый в данном случае:

```
FormatConditionsObject.Add(Type, Operator, Formula1, Formula2)
```

Обязательный параметр *Type* указывает на то, будет ли условный формат основан на значении, содержащемся в ячейке, или же на выражении. Он может принимать значения констант из следующего перечисления:

```
Enum xlFormatConditionType
    xlCellValue = 1      ' Значение
    xlExpression = 2     ' Выражение
End Enum
```

Когда параметр *Type* принимает значение xlCellValue, параметр *Operator* указывает на то, какой оператор будет использоваться вместе со значением. В противном случае параметр *Operator* игнорируется. Он может принимать значения констант из следующего перечисления:

```
Enum xlFormatConditionOperator
    xlBetween = 1        ' В промежутке
    xlNotBetween = 2     ' Вне промежутка
    xlEqual = 3          ' Равно
    xlNotEqual = 4       ' Не равно
    xlGreater = 5        ' Больше
    xlLess = 6           ' Меньше
    xlGreaterEqual = 7   ' Больше или равно
    xlLessEqual = 8      ' Меньше или равно
End Enum
```

Если параметр *Type* имеет значение xlCellValue, параметры *Formula1* и *Formula2* дают сравнительные значения. При этом применяется параметр *Operator* и содержащиеся в ячейке значения. Параметр *Formula2* используется только с константами xlBetween и xlNotBetween.

Например, рассмотрим код, задающий цвет заливки ячеек в диапазоне A1:C4 25% серого, если в ячейке содержится число от 0 до 10 включительно, и белый цвет в остальных случаях. Результат показан на рис. 19.24. Обратите внимание, что перед созданием нового объекта FormatConditions была произведена операция по снятию условного форматирования, которое могло присутствовать в указанном диапазоне.

```
Dim rng As Range
Dim i As Integer
Set rng = Range("A1:C4")
' Отмена ранее существовавшего форматирования
For i = rng.FormatConditions.Count To 1 Step -1
    rng.FormatConditions(i).Delete
```



```
Next
With rng
    .FormatConditions.Add xlCellValue, xlBetween, 0, 10
    .FormatConditions(1).Interior.Color = RGB(196, 196, 196)
    .FormatConditions.Add xlCellValue, xlNotBetween, 0, 10
    .FormatConditions(2).Interior.Color = RGB(255, 255, 255)
End With
```

	A	B	C
1	1		
2	2	3	100
3	-1		
4			
5			

Рис. 19.24. Вид диапазона после условного форматирования

Если параметр *Type* имеет значение *xlExpression*, параметр *Formula2* игнорируется, а *Formula1* дает формулу или выражение, задающее условие. Этот параметр может быть строкой, константой, ссылкой на ячейку или формулой. Для иллюстрации рассмотрим код, устанавливающий цвет заливки ячеек в зависимости от того, содержится ли в ячейках A1 и A2 одно и то же значение:

```
Dim rng As Range
Dim i As Integer
Set rng = Range("A1:A2")
' Отмена ранее существовавшего форматирования
For i = rng.FormatConditions.Count To 1 Step -1
    rng.FormatConditions(i).Delete
Next
With rng
    .FormatConditions.Add xlExpression, _
        Range("A1").Value = Range("A2").Value
    .FormatConditions(1).Interior.Color = _
        RGB(0, 0, 255)
    .FormatConditions.Add xlExpression, _
        Range("A1").Value <> Range("A2").Value
    .FormatConditions(2).Interior.Color = _
        RGB(255, 0, 0)
End With
```

Как показано в предыдущем примере, форматирование выполняется заданием неких свойств потомков объекта *FormatCondition*. В частности, свойства *Borders*, *Font* и *Interior* возвращают дочерние объекты с такими же именами, которые и определяют нужное пользователю форматирование.

Напоследок заметим, что существующий объект *FormatCondition* может быть удален с помощью метода *Delete* из семейства *FormatConditions*. А для его изменения используется метод *Modify* с синтаксисом:

```
FormatConditionObject.Modify(Type, Operator, Formula1, Formula2)
```

Смысл параметров в данном случае тот же самый, что и у метода *Add*.

Объект Interior

Объект *Interior* применяется для форматирования цвета и узора ячейки или диапазона. Он имеет только набор свойств, о которых мы поговорим в этом разделе.

Свойства Color и ColorIndex

Свойства Color и ColorIndex в данном случае аналогичны одноименным свойствам объекта Border, который рассматривался ранее. Они определяют цвет заливки ячеек.

Свойство Pattern

Свойство Pattern возвращает или задает рисунок внутри ячеек и может принимать значения констант из следующего перечисления:

```
Enum xlPattern
    xlPatternVertical = -4166      ' Вертикальная заштриховка
    xlPatternUp = -4162           ' Диагональные штрихи с наклоном вправо
    xlPatternNone = -4142         ' Отсутствие рисунка
    xlPatternHorizontal = -4128   ' Горизонтальная заштриховка
    xlPatternGray75 = -4126       ' 75% серого цвета
    xlPatternGray50 = -4125       ' 50% серого цвета
    xlPatternGray25 = -4124       ' 25% серого цвета
    xlPatternDown = -4121         ' Диагональные штрихи с наклоном влево
    xlPatternAutomatic = -4105    ' Заливка серым цветом
    xlPatternSolid = 1            ' Сплошная заливка
    xlPatternChecker = 9          ' Шахматное поле
    xlPatternSemiGray75 = 10      ' Пятна 75%-ного серого цвета
    xlPatternLightHorizontal = 11 ' Толстые горизонтальные серые полосы
    xlPatternLightVertical = 12   ' Толстые вертикальные серые полосы
    xlPatternLightDown = 13       ' Диагональные штрихи с наклоном влево
    xlPatternLightUp = 14         ' Диагональные штрихи с наклоном вправо
    xlPatternGrid = 15            ' Сетка
    xlPatternCrissCross = 16       ' Крестообразная заливка
    xlPatternGray16 = 17          ' Оттенок серого
    xlPatternGray8 = 18           ' Оттенок серого
End Enum
```

Обратите внимание на то, что это свойство является дополнительным способом доступа к заливке в оттенках серого.

Свойства PatternColor и PatternColorIndex

Свойства PatternColor и PatternColorIndex задают цвет узора, заполняющего ячейки диапазона. Дополнительную информацию о цветах и их индексах можно найти в разделе, посвященном свойствам Color и ColorIndex объекта Border.

Объекты PivotField, PivotItem и PivotTable

Объекты PivotField, PivotItem и PivotTable будут подробно рассматриваться в главе 20, посвященной сводным таблицам.

Объект QueryTable

Объект QueryTable представляет собой таблицу, построенную на основе данных из внешнего источника, например сервера Microsoft SQL или базы данных Microsoft Access. В этой книге мы не будем подробно обсуждать объект QueryTable, так как существуют более удобные способы получить данные из внешнего источника.

Объект Validation

Объект Validation позволяет контролировать значения, вводимые пользователем в ячейки, устанавливать допустимые значения, интервалы и тип данных, а также создавать приглашения на ввод или сообщение об ошибке при вводе неверных данных. Имейте в виду, что семейства *Validations* не существует.

Процесс проверки данных состоит из трех частей: сама проверка, приглашение на ввод данных, появляющееся при активизации ячейки, и сообщение об ошибке, которое появляется при вводе некорректных данных.

Объект Validation обладает методами: *Add*, *Delete* и *Modify*. Для проверки диапазона используется метод *Add* со следующим синтаксисом:

```
ValidationObject.Add(Type, AlertStyle, Operator, Formula1, Formula2)
```

Обратите внимание на сходство между параметрами метода *Add* объектов Validation и *FormatCondition*.

Обязательный параметр *Type* может принимать значения констант из следующего перечисления:

```
Enum XlDVType
    xlValidateInputOnly = 0      ' Отсутствие проверки
    xlValidateWholeNumber = 1    ' Проверка целых чисел
    xlValidateDecimal = 2       ' Проверка вещественных чисел
    xlValidateList = 3          ' Проверка списков
    xlValidateDate = 4          ' Проверка дат
    xlValidateTime = 5          ' Проверка времени
    xlValidateTextLength = 6     ' Проверка длины текста
    xlValidateCustom = 7        ' Проверка по пользовательскому критерию
End Enum
```

Если параметр *Type* имеет значение *xlValidateInputOnly*, приложение считает все данные корректными по умолчанию. Это значение используется в тех случаях, когда требуется приглашение на ввод, о котором мы поговорим чуть позже.

Необязательный параметр *AlertStyle* указывает, какие кнопки будут присутствовать в диалоговом окне с сообщением об ошибке, которое появляется при вводе некорректных данных. Он может принимать значения констант из следующего перечисления:

```
Enum XlDAlertStyle
    xlValidAlertStop = 1        ' Кнопки OK и Cancel
    xlValidAlertWarning = 2     ' Кнопки Retry и Cancel
    xlValidAlertInformation = 3 ' Кнопки Yes, No и Cancel
End Enum
```

Необязательный параметр *Operator* представляет собой оператор, используемый при проверке корректности данных, и может принимать значения констант из следующего перечисления:

```
Enum XlFormatConditionOperator
    xlBetween = 1              ' В промежутке
    xlNotBetween = 2           ' Вне промежутка
    xlEqual = 3                ' Равно
    xlNotEqual = 4             ' Не равно
    xlGreater = 5              ' Больше
    xlLess = 6                 ' Меньше
    xlGreaterEqual = 7         ' Больше или равно
    xlLessEqual = 8            ' Меньше или равно
End Enum
```


Параметр *Formula1* определяет первую часть условия для проверки данных, а если параметр *Operator* имеет значение *xlBetween* или *xlNotBetween*, параметр *Formula2* определяет вторую часть проверочного условия.

Чтобы лучше понять этот достаточно сложный объект, взглянем на соответствующее диалоговое окно, которое используется в пользовательском интерфейсе Excel. На рис. 19.25 оно открыто на вкладке Параметры (Settings).

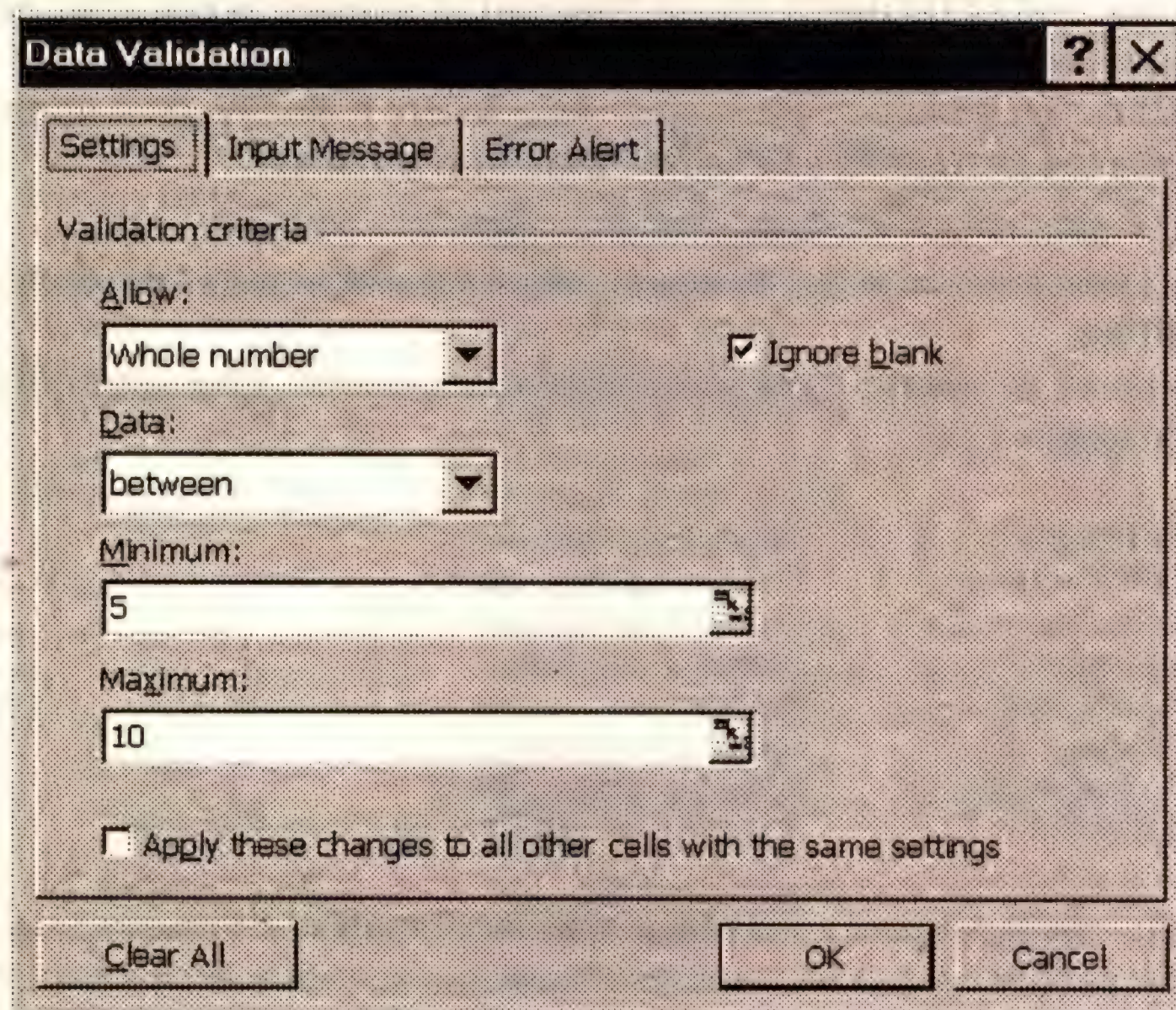


Рис. 19.25. Диалоговое окно Проверка вводимых значений, открытое на вкладке Параметры

Это диалоговое окно соответствует настройкам:

```
Type:=xlValidateWholeNumber
Operator:=xlBetween
Formula1:="5"
Formula2:="10"
IgnoreBlank = True
```

Дополнительную информацию о значениях параметра *Type* можно получить, щелкнув на кнопке со знаком ?, расположенной в верхнем правом углу диалогового окна Data Validation (Проверка вводимых значений), и затем исследовав раскрывающийся список Allow (Тип данных). Помните, что элементы управления на вкладке Settings (Параметры) изменяются в зависимости от значения, выбранного в раскрывающемся списке Allow (Тип данных).

Вкладка Input Message (Сообщение для ввода) показана на рис. 19.26. Значения в этом диалоговом окне соответствуют следующим свойствам объекта Validation:

```
ShowInput = True
InputTitle = "Ввод:"
InputMessage = "Введите число"
```

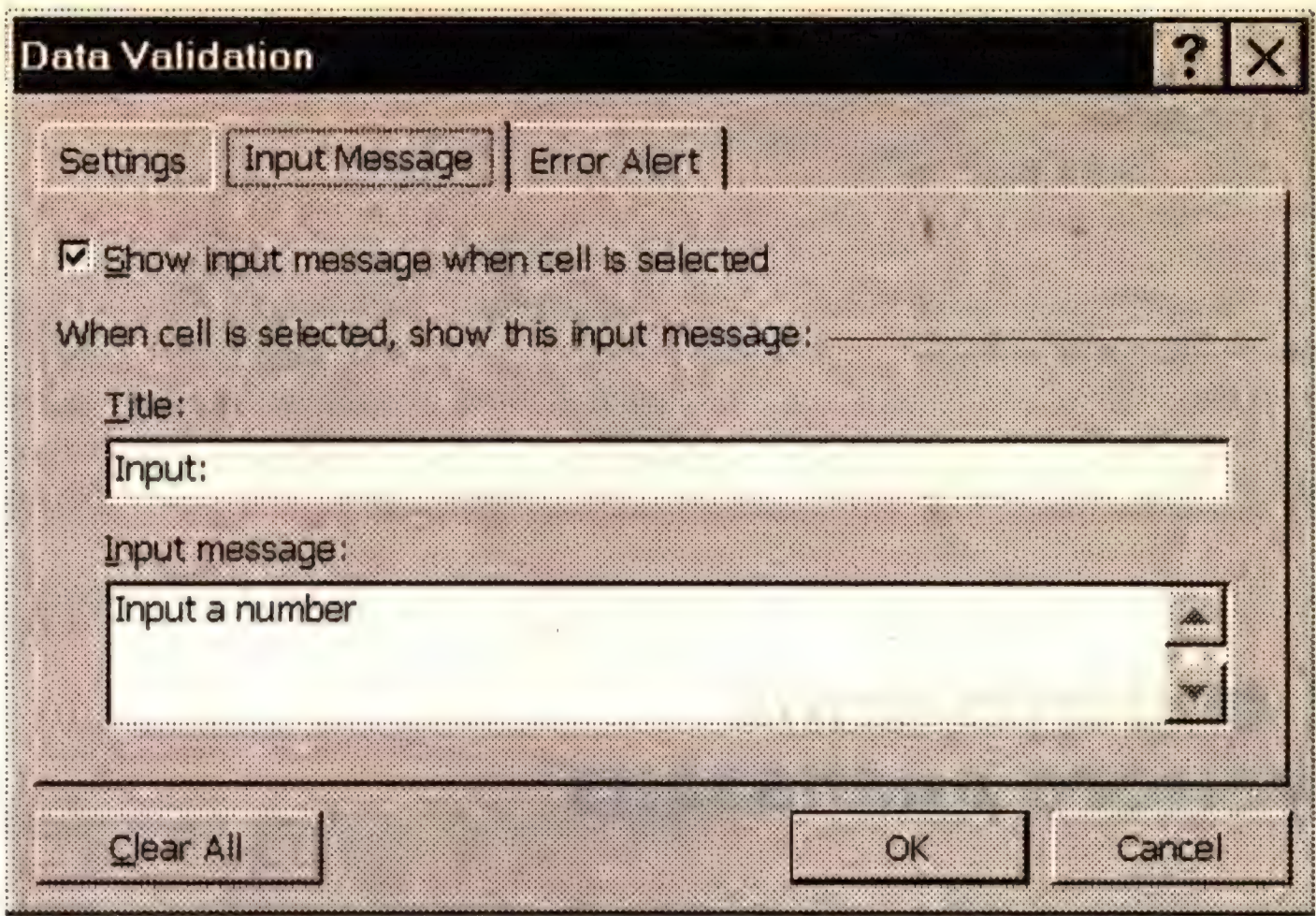



Рис. 19.26. Вкладка Сообщение для ввода диалогового окна Проверка вводимых значений

На рис. 19.27 показана вкладка Error Alert (Сообщение об ошибке). Это диалоговое окно соответствует следующим свойствам объекта Validation:

```
ShowError = True
ErrorTitle = "Ошибка:"
ErrorMessage = "Это ошибка"
```

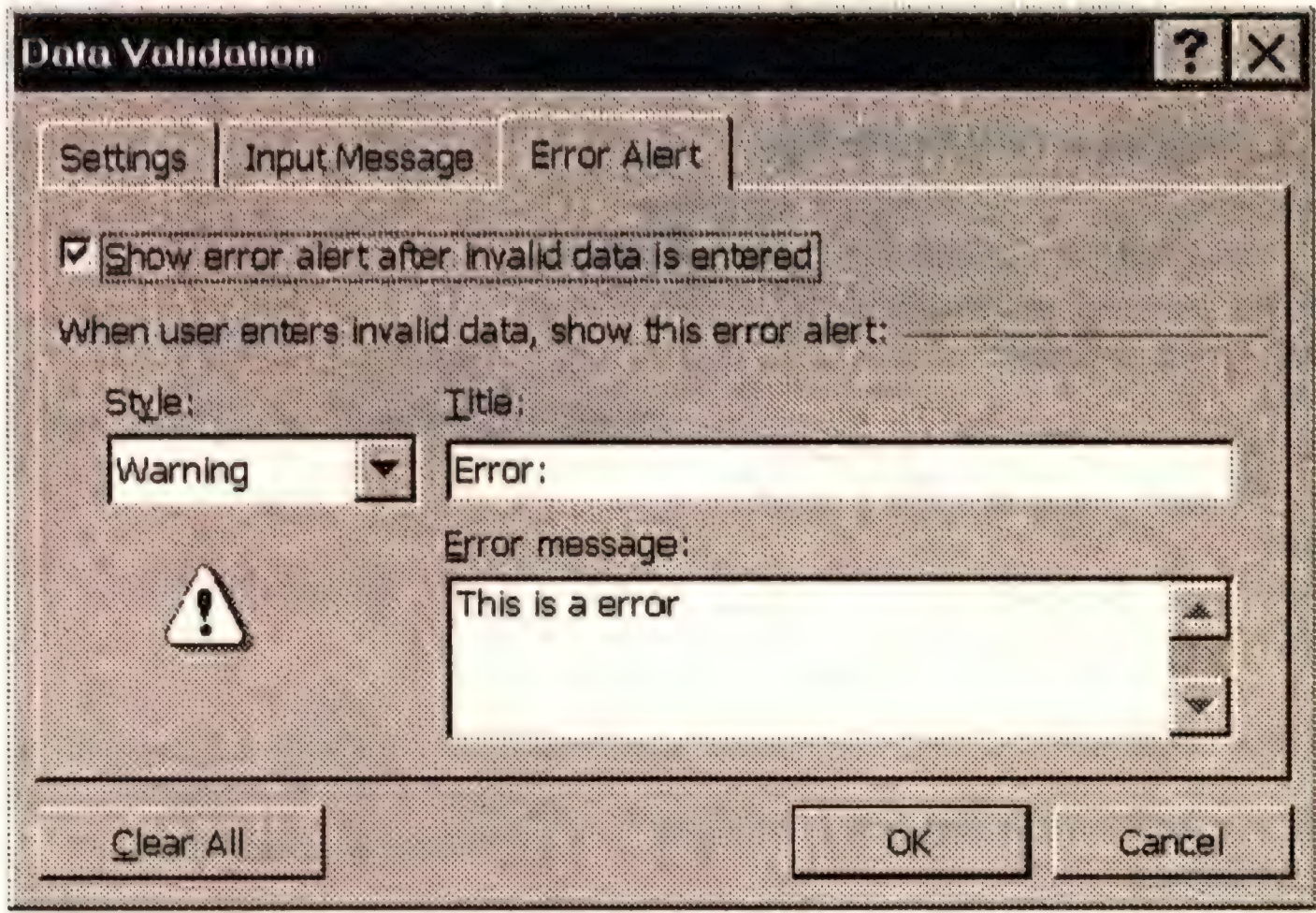


Рис. 19.27. Вкладка Сообщение об ошибке диалогового окна Проверка вводимых значений

Теперь можно собрать все куски воедино и показать, как настраивается проверка данных для диапазона ячеек. Учтите, что первым делом требуется удалить старые критерии проверки.

```
With Range("A1:D4").Validation
.Delete
```



```

        .Add Type:=xlValidateWholeNumber, _
        AlertStyle:=xlValidAlertStop, _
        Operator:=xlBetween, _
        Formula1:="5", Formula2:="10"
    .IgnoreBlank = True
    .ShowInput = True
    .InputTitle = "Ввод:"
    .InputMessage = "Введите число"
    .ShowError = True
    .ErrorTitle = "Ошибка:"
    .ErrorMessage = "Это ошибка"
End With

```

Пример: определение используемого диапазона

Как упоминалось в главе 18, метод `UsedRange` не всегда дает корректные результаты. Иногда он неверно возвращает используемый в данный момент диапазон, то есть наименьшую прямоугольную область, которая содержит ячейки с данными. В любом случае, если у вас были проблемы с методом `UsedRange`, используйте вместо него данную в этом разделе функцию.

Операция `GetUserRange` достаточно проста. Ее код дан в листинге 19.1. Для начала определяется координата (номер строки и столбца) верхнего левого и нижнего правого углов используемого диапазона, а затем этот диапазон пытаются сократить, если он содержит пустые строки или столбцы. В последнем случае используется функция `CountA`, подсчитывающая количество непустых ячеек.

Листинг 19.1. Функция `GetUsedRange`

```

Function GetUsedRange(ws As Worksheet) As Range
    ' Предположим, что свойство UsedRange дает расширенный
    ' диапазон вместо реально используемого.
    Dim s As String, x As Integer
    Dim rng As Range
    Dim r1Fixed As Integer, c1Fixed As Integer
    Dim r2Fixed As Integer, c2Fixed As Integer
    Dim i As Integer
    Dim r1 As Integer, c1 As Integer
    Dim r2 As Integer, c2 As Integer

    Set GetUsedRange = Nothing

    ' Начнем с используемого приложением диапазона
    Set rng = ws.UsedRange

    ' Определение граничных ячеек для используемого диапазона
    ' Это ячейки от Cells(r1,c1) до Cells(r2,c2)
    r1 = rng.Row
    r2 = rng.Rows.Count + r1 - 1
    c1 = rng.Column
    c2 = rng.Columns.Count + c1 - 1

    ' Сохранение существующих значений

```



```
r1Fixed = r1
c1Fixed = c1
r2Fixed = r2
c2Fixed = c2

' Поиск пустых строк перебором сверху вниз
' При обнаружении пустой строки она удаляется
For i = 1 To r2Fixed - r1Fixed + 1
    If Application.CountA(rng.Rows(i)) = 0 Then
        ' Пустая строка удаляется
        r1 = r1 + 1
    Else
        ' При обнаружении непустой строки выход из цикла
        Exit For
    End If
Next

' Процесс повторяется для столбцов, слева направо
For i = 1 To c2Fixed - c1Fixed + 1
    If Application.CountA(rng.Columns(i)) = 0 Then
        c1 = c1 + 1
    Else
        Exit For
    End If
Next

' Возвращение исходного диапазона
Set rng = _
    ws.Range(ws.Cells(r1, c1), ws.Cells(r2, c2))

' Повторный процесс
r1Fixed = r1
c1Fixed = c1
r2Fixed = r2
c2Fixed = c2

' Проверка строк снизу вверх
For i = r2Fixed - r1Fixed + 1 To 1 Step -1
    If Application.CountA(rng.Rows(i)) = 0 Then
        r2 = r2 - 1
    Else
        Exit For
    End If
Next

' Проверка столбцов справа налево
For i = c2Fixed - c1Fixed + 1 To 1 Step -1
    If Application.CountA(rng.Columns(i)) = 0 Then
        c2 = c2 - 1
    Else
        Exit For
    End If
Next

Set GetUsedRange = _
    ws.Range(ws.Cells(r1, c1), ws.Cells(r2, c2))

End Function
```


Пример: выделение ячеек по заданному признаку

В пользовательском интерфейсе Excel нет встроенного метода выделения ячеек листа на основе различных критериев. Например, не существует возможности выделить все ячейки, значения которых принадлежат интервалу от 0 до 100, или ячейки, содержащие дату после 1 января 1998. Также невозможно выделить только те ячейки в столбце, значения которых отличаются от значения предшествующей ячейки. Последнее обстоятельство крайне полезно при наличии отсортированного столбца, из которого требуется извлечь уникальные значения, как показано на рис. 19.28.

Создадим небольшую служебную программу, добавив ее к приложению SRXUtils, которая позволяет выделять ячейки по простому критерию. Впоследствии к ней можно добавить дополнительные критерии выделения.

Для начала требуется добавить в лист DataSheet для надстройки SRXUtils еще одну строку, как показано на рис. 19.29.

12	Denver
13	Denver
14	Denver
15	Denver
16	Los Angeles
17	Los Angeles
18	New York
19	New York
20	New York
21	New York
22	New York
23	Portland
24	Portland
25	Seattle
26	Seattle

Рис. 19.28. Выделение уникальных значений

	A	B	C	D	E	F	G	H
1	Utility	OnAction Proc	Procedure	In Workbook	Menu Item	SubMenu Item	On Wks Menu	On Chart Menu
2	Activate Sheet	RunUtility	ActivateSheet	ThisWorkbook	&Activate Sheet		TRUE	TRUE
3	Print Charts	RunUtility	PrintCharts	Print.utl	&Print	Embedded &Charts	TRUE	TRUE
4	Print Pivot Tables	RunUtility	PrintPivotTables	Print.utl		&Pivot Tables	TRUE	TRUE
5	Print Sheets	RunUtility	PrintSheets	Print.utl		&Sheets	TRUE	TRUE
6	Select Special	RunUtility	SelectSpecial	ThisWorkbook	S&elect Special		TRUE	FALSE
7	Sort Sheets	RunUtility	SortSheets	ThisWorkbook	&Sort Sheets		TRUE	TRUE

Рис. 19.29. Дополнение листа DataSheet

Конструирование служебной программы

Для сохранения относительной простоты служебной программы используем только следующие критерии поиска:

- выделить, если значение предыдущей ячейки отличается;
- выделить, если значение предыдущей ячейки идентично;
- выделить, если ячейка пуста;
- выделить, если ячейка не пуста.

В качестве диапазона поиска по данным критериям будет использоваться выделенный диапазон активного листа. Имейте в виду, что при этом требуется проверить, чтобы на листе был выделен именно диапазон ячеек, а не, например, диаграмма. Для поиска по первым двум критериям нужно, чтобы данные были введены в один столбец или в одну строку или же в часть того или другого. Поиск по двум остальным критериям может осуществляться в диапазоне произвольной формы.

Для удобства пользователя, если выделенный диапазон состоит из одной ячейки, служебная программа определит пользовательский диапазон для последних двух критериев и для пользовательской части столбца, содержащего активную ячейку, по первым двум критериям.

Дополнительно будет введена функция увеличения текущего выделенного диапазона, включая всю строку или весь столбец. Например, если применить поиск к листу, показанному на рис. 19.28, будут выделены строки 12, 16, 18, 23 и 25.

Конструирование диалогового окна

Теперь, когда план действий ясен, пришло время сконструировать диалоговое окно, показанное на рис. 19.30. Это объект UserForm с именем `dlgSelectSpecial`. Его свойство `Caption` должно иметь значение **Выделение по признаку**.

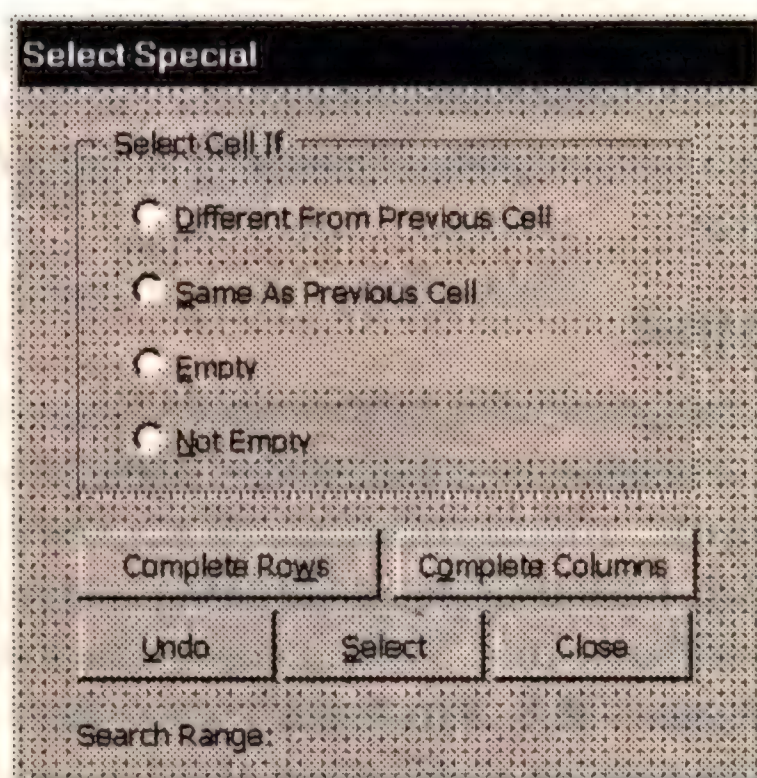


Рис. 19.30. Диалоговое окно Выделение по признаку

Пользователь вручную устанавливает переключатель **Выделить ячейки, если**. Диапазон поиска показывается в нижней части диалогового окна.

Процесс создания подобных форм имеет ряд особенностей. Внимательно прочитайте их, прежде чем приступить к конструированию.

Элемент управления Frame

Элемент управления `Frame` используется для группировки прочих элементов управления, имеющих одинаковое назначение. Однако в случае с переключателем он

имеет более глубокий смысл. Дело в том, что кнопки переключателя, объединенные рамкой, являются взаимоисключающими. То есть если пользователь выделяет одну из кнопок, с остальных автоматически снимается выделение.

Чтобы гарантировать наличие кнопок переключателя внутри рамки, а не просто поверх нее, убедитесь в том, что в момент щелчка на кнопке `OptionButton` в диалоговом окне `Toolbox` рамка выделена. Затем создайте кнопку переключателя внутри рамки. Если вы решите получить остальные кнопки переключателя методом копирования и вставки, убедитесь в том, что при выборе команды `Paste` рамка выделена.

Имена элементов управления

Имена элементов управления были выбраны в соответствии с моим стандартом именования:

- `fraType (frame)`
- `optDifferent`
- `optSame`
- `optEmpty`
- `cmdSelect`
- `cmdCancel`
- `cmdUndo`
- `cmdCompleteRows`
- `cmdCompleteColumns`
- `lblSearchRange`

В данном случае не потребуется задавать многочисленные свойства `Name` и `Accelerator`. Присвойте ярлыку `lblSearchRange` свойства `WordWrap` значение `False`, чтобы подпись занимала только одну строку. Присвойте также свойству `TabStop` объекта `lblSearchRange` значение `False`.

Порядок активизации

При конструировании нестандартного диалогового окна важно правильно указать порядок активизации всех элементов управления. Вряд ли стоит заставлять пользователя бегать глазами по форме, отслеживая, какой элемент активизировался при очередном нажатии клавиши `Tab`. Проще всего в данном случае воспользоваться диалоговым окном `Tab Order`, показанном на рис. 19.31. Для его вызова нужно выбрать команду `Tab Order` в меню `View` редактора VBA. Это диалоговое окно позволяет не только увидеть текущий порядок активизации элементов, но и изменять его нужным образом.

Помните, что элемент управления со значением свойства `TabIndex`, равным нулю, будет активен при первом запуске сконструированной формы. Воспользоваться диалоговым окном `Tab Order` придется два раза: при выделенной форме и при выделенной рамке, так как вам потребуется указать порядок активизации кнопок переключателя.

Заключительные советы

Нужно отметить, что с помощью команд меню `Format` редактора VBA можно легко выравнивать элементы управления и поменять их размер, что позволяет придать форме более профессиональный вид. Другим способом добиться единообразия является копирование и вставка элементов управления.

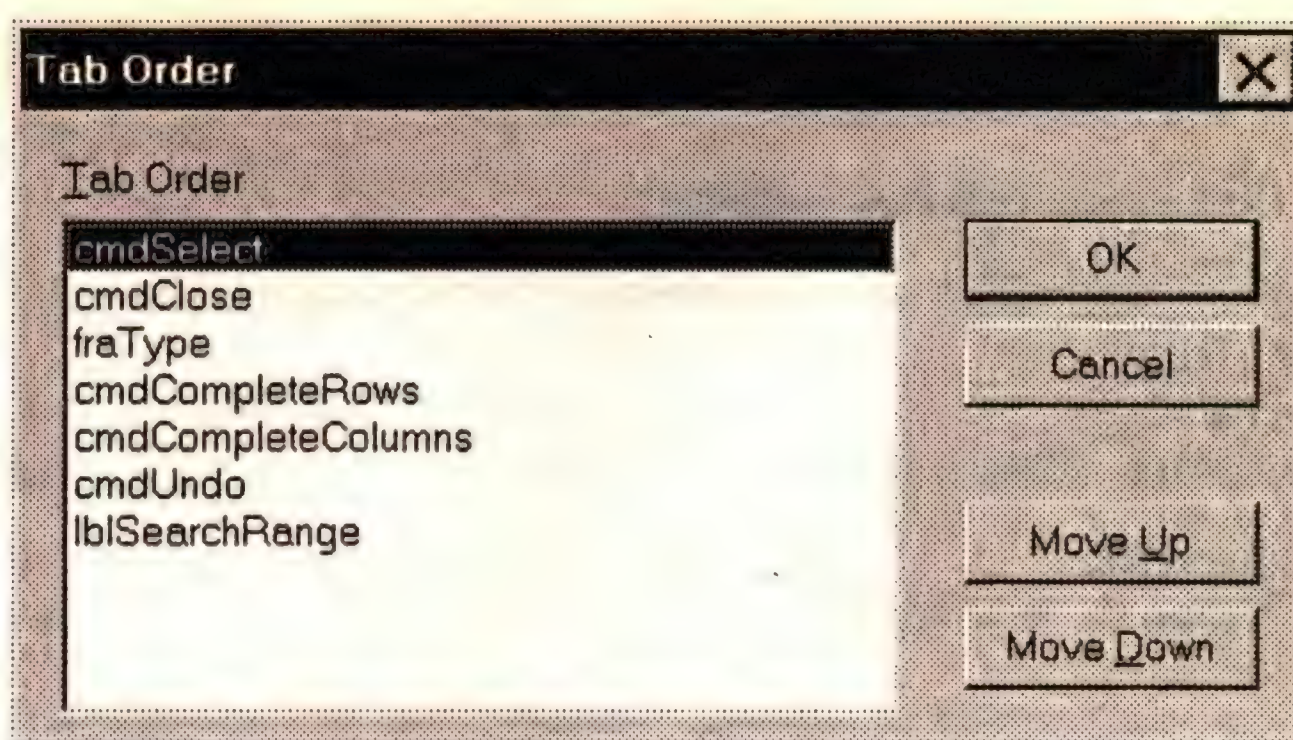


Рис. 19.31. Диалоговое окно TabOrder

Написание кода

Теперь, когда диалоговое окно создано, пришло время сопоставить с ним код.

В стандартном модуле `basMain` поместим код, соответствующий диалоговому окну Выделение по признаку. Однако следует учитывать возможность выделения на активном листе элемента, отличного от набора ячеек, например рисунка или диаграммы. В этом случае вместо диалогового окна требуется показать информационное окно с сообщением о неподходящем выделенном наборе. Соответствующий код приводится в листинге 19.2.

Листинг 19.2. Процедура `SelectSpecial`

```
Sub SelectSpecial()  
    ' Проверка корректности выделенного набора  
    If TypeName(Selection) <> "Range" Then  
        MsgBox "Требуется выделить диапазон ячеек.", vbCritical  
    Else  
        dlgSelectSpecial.Show  
    End If  
  
End Sub
```

Обратите внимание на использование функции `TypeName`. Будучи примененной к объекту:

`TypeName(ObjectVariable),`

она возвращает имя этого объекта.

Теперь требуется сделать несколько описаний на уровне модуля, показанных в листинге 19.3.

Листинг 19.3. Описание уровня модуля `dlgSelectSpecial`

```
Option Explicit  
' Эти переменные используются в нескольких процедурах  
Dim rngSearch As Range  
Dim rngForUndo As Range
```

Событие `Initialize` формы используется для инициализации элементов управления. Как показано в листинге 19.4, сначала требуется заблокировать некоторые

командные кнопки и написать название подписи lblSearchRange. Здесь же можно указать переменные уровня модуля.

Листинг 19.4. Процедура события Initialize

```
Private Sub UserForm_Initialize()  
  
cmdSelect.Enabled = False  
cmdUndo.Enabled = False  
lblSearchRange.Caption = "Область поиска: Отсутствует."  
  
Set rngSearch = Selection  
Set rngForUndo = rngSearch  
  
End Sub
```

Кнопка Заккрыть просто выгружает форму. Соответствующий код показан в листинге 19.5.

Листинг 19.5. Процедура события cmdClose_Click

```
Private Sub cmdClose_Click()  
    Unload Me  
End Sub
```

Кстати, проверить работу имеющихся фрагментов программы можно, запустив событие Initialize. Достаточно поместить курсор на это событие и нажать клавишу F5.

Кнопка Отменить возвращает выделенный набор в первоначальное состояние, которое сохраняется в переменной уровня модуля rngForUndo. Ее код показан в листинге 19.6.

Листинг 19.6. Процедура события cmdUndo_Click

```
Private Sub cmdUndo_Click()  
    If Not rngForUndo Is Nothing Then  
        rngForUndo.Select  
        cmdUndo.Enabled = False  
    End If  
End Sub
```

Первое, что сделает пользователь после появления диалогового окна, — выберет положение переключателя в верхней рамке, чем определит критерий поиска. Некоторые варианты выбора требуют наличия ограниченного диапазона. Для реакции на выбор пользователя применяется процедура GetSearchRange. Код, запускающий эту процедуру для каждого положения переключателя, показан в листинге 19.7.

Листинг 19.7. Процедура обработки событий для кнопок переключателя

```
Private Sub optDifferent_Click()  
    GetSearchRange  
End Sub  
  
Private Sub optEmpty_Click()  
    GetSearchRange  
End Sub  
  
Private Sub optNotEmpty_Click()  
    GetSearchRange
```



```
End Sub
```

```
Private Sub optSame_Click()  
    GetSearchRange  
End Sub
```

Код самой процедуры GetSearchRange приводится в листинге 19.8.

Листинг 19.8. Процедура GetSearchRange

```
Private Sub GetSearchRange()  
  
    ' Выбор диапазона поиска на основе указанного критерия.  
    ' При выборе вариантов Отличный или Одинаковый проверка диапазона  
    ' При выборе одной ячейки:  
    ' - Для совпадений Отличный или Одинаковый использовать столбец  
    ' - Для совпадений Пустая или Непустая использовать строку  
    ' Мы знаем, что rngSearch – это диапазон ячеек.  
    ' Если выделен некорректный диапазон, кнопка Выделить блокируется.  
  
    Dim cColumns As Integer, cRows As Integer  
  
    cmdSelect.Enabled = True ' Значение может быть временным  
  
    If optDifferent Or optSame Then  
        ' Диапазон поиска должен быть  
        ' одним столбцом или строкой или их частью.  
  
        cColumns = rngSearch.Columns.Count  
        cRows = rngSearch.Rows.Count  
  
        If rngSearch.Areas.Count > 1 Or  
        (cColumns <> 1 And cRows <> 1) Then  
            lblSearchRange.Caption = "Требуется один столбец или строка или их часть."  
            cmdSelect.Enabled = False  
            Exit Sub  
        End If  
  
        ' Одна ячейка расширяется до используемой части столбца  
        If cColumns = 1 And cRows = 1 Then  
            Set rngSearch = Application.Intersect(  
                rngSearch.EntireColumn, ActiveSheet.UsedRange)  
        End If  
  
    ElseIf optEmpty Or optNotEmpty Then  
        ' Если выделена одна ячейка, она расширяется до используемого диапазона.  
        If rngSearch.Cells.Count = 1 Then  
            Set rngSearch = ActiveSheet.UsedRange  
        End If  
  
    End If  
  
    lblSearchRange.Caption = "Диапазон поиска: " &  
        rngSearch.Address(RowAbsolute:=False, ColumnAbsolute:=False)  
  
End Sub
```

Когда пользователь нажимает клавишу Выделить, начинается поиск, основанный на выбранном критерии. То есть требуется вызов соответствующей процедуры.

После нового выбора кнопка Выделить блокируется. Но так как функции CompleteRows и CompleteColumns до сих пор доступны, закрывать основное диалоговое окно не требуется. Код, запускаемый при нажатии кнопки Выделить, показан в листинге 19.9.

Листинг 19.9. Процедура события cmdSelect_Click

```
Private Sub cmdSelect_Click()

' Проверка выбранного пользователем переключателя
' и вызов соответствующей процедуры
If optDifferent Then
    SelectIfDifferent
ElseIf optSame Then
    SelectIfSame
ElseIf optEmpty Then
    SelectIfEmpty
ElseIf optNotEmpty Then
    SelectIfNotEmpty
End If

cmdSelect.Enabled = False

End Sub
```

Код процедуры SelectIfDifferent показан в листинге 19.10. Она производит поиск в диапазоне rngSearch, отмечая ячейки, содержимое которых отличается от содержимого их предшественниц. Так как заранее неизвестно, что именно будет фигурировать в качестве диапазона поиска: столбец или строка (это может быть даже их часть), проще всего воспользоваться двойным циклом For. Однако более эффективно разбить код на две части (cColumns=1 и cRows=1). На первую ячейку требуется обратить особое внимание, так как она должна быть включена в выделенный набор, который содержится в объектной переменной Range с именем rngMatch. Это включение осуществляется с помощью функции Union.

Кроме того, всегда следует помнить о том, что переменная rngMatch может иметь значение Nothing. В этом случае функция Union также вернет значение Nothing. Другими словами:

```
Application.Union(Something, Nothing) = Nothing
```

Листинг 19.10. Процедура SelectIfDifferent

```
Private Sub SelectIfDifferent()

Dim rngMatch As Range
Dim vCellValue As Variant
Dim vPreviousCellValue As Variant
Dim cMatches As Integer
Dim oCell As Object
Dim cRows As Integer, cColumns As Integer
Dim r As Integer, c As Integer

' Узнаем число строк и столбцов
cColumns = rngSearch.Columns.Count
cRows = rngSearch.Rows.Count

' Начало поиска
cMatches = 0
Set rngMatch = Nothing
```



```

For r = 1 To cRows
    For c = 1 To cColumns

        Set oCell = rngSearch.Cells(r, c)
        vCellValue = oCell.Value
        vCellValue = CStr(vCellValue)

        If r = 1 And c = 1 Then
            ' Включение первой ячейки
            If rngMatch Is Nothing Then
                Set rngMatch = oCell
            Else
                Set rngMatch = Application.Union(rngMatch, oCell)
            End If
            cMatches = cMatches + 1
            ' Сохраняем значение для следующего сравнения
            vPreviousCellValue = vCellValue
        Else
            ' Сравниваем с предыдущей ячейкой
            vCellValue = rngSearch.Cells(r, c).Value
            vCellValue = CStr(vCellValue)
            If vCellValue <> vPreviousCellValue Then
                If rngMatch Is Nothing Then
                    Set rngMatch = oCell
                Else
                    Set rngMatch = Application.Union(rngMatch, oCell)
                End If
                cMatches = cMatches + 1
            End If
            ' Сохраняем значение для следующего сравнения
            vPreviousCellValue = vCellValue
        End If
    Next ' Столбец
Next ' Строка

' Выделение диапазона
If cMatches > 0 Then
    rngMatch.Select
    cmdUndo.Enabled = False
Else
    MsgBox "Нет совпадающих ячеек. Выделенный набор не меняется.", vbInformation
    cmdUndo.Enabled = False
End If

End Sub

```

Процедура **SelectIfSame**, код которой показан в листинге 19.11, аналогична процедуре **SelectIfDifferent**. Основное различие состоит в отсутствии необходимости включать в выделенный набор первую ячейку.

Листинг 19.11. Процедура **SelectIfSame**

```

Private Sub SelectIfSame()

    Dim rngMatch As Range
    Dim vCellValue As Variant
    Dim vPreviousCellValue As Variant
    Dim cMatches As Integer
    Dim oCell As Object
    Dim cRows As Integer, cColumns As Integer

```

продолжение ➤

Листинг 19.11 (продолжение)

```

Dim r As Integer, c As Integer

' Узнаем число строк и столбцов
cColumns = rngSearch.Columns.Count
cRows = rngSearch.Rows.Count

' Начало поиска
cMatches = 0
Set rngMatch = Nothing

For r = 1 To cRows
    For c = 1 To cColumns

        Set oCell = rngSearch.Cells(r, c)
        vCellValue = oCell.Value
        vCellValue = CStr(vCellValue)

        If r = 1 And c = 1 Then
            ' Сохраним первое значение для последующего сравнения
            vPreviousCellValue = vCellValue
        Else
            ' Сравнение с предыдущей ячейкой
            vCellValue = rngSearch.Cells(r, c).Value
            vCellValue = CStr(vCellValue)
            If vCellValue = vPreviousCellValue Then
                If rngMatch Is Nothing Then
                    Set rngMatch = oCell
                Else
                    Set rngMatch = Application.Union(rngMatch, oCell)
                End If
                cMatches = cMatches + 1
            End If
            ' Сохранение значения для следующего сравнения.
            vPreviousCellValue = vCellValue
        End If
    Next ' Столбец
Next ' Строка

' Выделение диапазона.
If cMatches > 0 Then
    rngMatch.Select
    cmdUndo.Enabled = False
Else
    MsgBox "Нет совпадающих ячеек. Выделенный набор не меняется.", vbInformation
    cmdUndo.Enabled = False
End If

End Sub

```

Процедуры `SelectIfEmpty` и `SelectIfNotEmpty` практически идентичны. Первая из них показана в листинге 19.12.

Листинг 19.12. Процедура `SelectIfEmpty`

```

Private Sub SelectIfEmpty()

Dim rngMatch As Range
Dim cMatches As Integer
Dim oCell As Object

```



```
Dim cRows As Integer, cColumns As Integer
Dim r As Integer, c As Integer

' Узнаем число столбцов и строк
cColumns = rngSearch.Columns.Count
cRows = rngSearch.Rows.Count

' Начало поиска
cMatches = 0
Set rngMatch = Nothing

For r = 1 To cRows
    For c = 1 To cColumns

        Set oCell = rngSearch.Cells(r, c)

        If IsEmpty(oCell) Then
            If rngMatch Is Nothing Then
                Set rngMatch = oCell
            Else
                Set rngMatch = Application.Union(rngMatch, oCell)
            End If
            cMatches = cMatches + 1
        End If

    Next ' Столбец
Next ' Строка

' Выделение диапазона
If cMatches > 0 Then
    rngMatch.Select
    cmdUndo.Enabled = False
Else
    MsgBox " Нет совпадающих ячеек. Выделенный набор не меняется.", vbInformation
    cmdUndo.Enabled = False
End If

End Sub
```

Чтобы определить процедуру `SelectIfNotEmpty`, достаточно изменить строку:

```
If IsEmpty(oCell) Then
```

строкой:

```
If Not IsEmpty(oCell) Then
```

И наконец, процедуры `CompleteRows` и `CompleteColumns`, вызываемые при наступлении события `Click` для кнопок `Заполнить строки` и `Заполнить столбцы`, также практически идентичны. Код процедуры `CompleteColumns` показан в листинге 19.13.

Листинг 19.13. Процедура `cmdCompleteColumns_Click`

```
Private Sub cmdCompleteColumns_Click()
```

```
' Для каждой выбранной ячейки выделяется весь столбец
```

```
Dim oCell As Object
```

```
Dim rngNew As Range
```

```
Set rngNew = Nothing
```

продолжение ➤

Листинг 19.13 (продолжение)

```
For Each oCell In Selection
    If rngNew Is Nothing Then
        Set rngNew = oCell.EntireColumn
    Else
        Set rngNew = Union(rngNew, oCell.EntireColumn)
    End If
Next

rngNew.Select
cmdUndo.Enabled = True

End Sub
```

Для получения кода процедуры CompleteRows достаточно заменить объект EntireColumn объектом EntireRow.

20 Сводные таблицы

В этой главе мы поговорим о сводных таблицах, а также о способах их создания и форматирования программным образом.

Сводные таблицы

Так как предполагается, что читатели этой книги имеют основные навыки работы в Excel, обзор самого понятия сводных таблиц будет достаточно поверхностным.

Один из самых мощных инструментов Excel — *сводные таблицы (Pivot tables)* — разработан для выполнения трех основных задач:

- импорт внешних данных;
- группировка данных, например вычисление суммы или среднего;
- отображение данных нестандартными способами.

Сводные таблицы формируются на основе данных как из внешних источников, так и из таблиц Excel. Например, сводную таблицу можно построить на основе базы Access. Однако импорт внешних данных осуществляется только при наличии на компьютере пользователя соответствующих драйверов. Кроме того, в Excel существует ряд ограничений на импорт данных с помощью сводных таблиц. К примеру, длина строк не должна превышать 255 символов, что затрудняет использование SQL в качестве источника данных.

В конечном счете импорт данных с помощью сводных таблиц является достаточно проблематичным делом. Намного проще импортировать нужные данные непосредственно на лист Excel (с помощью таких методов, как, к примеру, DAO или GetRows) и уже затем создать сводную таблицу. Именно поэтому наше обсуждение ограничится созданием сводных таблиц на основе данных из Excel.

В табл. 20.1 перечислены данные по продажам воображаемой сети ресторанов быстрого питания, которая имеет как собственные магазины (company), так и магазины посторонних фирм, торгующих по их лицензии и под их маркой (franchise). Они будут использованы для построения сводной таблицы. Источником данных служит лист Excel, который дважды содержит номера строк, как в табл. 20.1. Дополнительные строки предназначены для аналогичных данных за 1997 год. То есть первый столбец в остальной части таблицы содержит число 1997.

Таблица 20.1. Исходные данные для сводной таблицы (за 1998 год)

Год	Период	Код магазина	Город	Тип магазина	Число сделок	Продажи
1998	1	BO-1	Бостон	Company	3881	\$6,248.00
1998	1	BO-2	Бостон	Company	3789	\$5,722.00
1998	1	BO-3	Бостон	Company	3877	\$6,278.00
1998	1	BO-4	Бостон	Company	3862	\$6,123.00
1998	1	BO-5	Бостон	Franchise	4013	\$6,861.00
1998	1	BO-6	Бостон	Franchise	3620	\$5,039.00
1998	2	BO-1	Бостон	Company	3948	\$6,468.00
1998	2	BO-2	Бостон	Company	3878	\$6,301.00
1998	2	BO-3	Бостон	Company	3911	\$6,390.00
1998	2	BO-4	Бостон	Company	3926	\$6,438.00
1998	2	BO-5	Бостон	Franchise	3990	\$6,767.00
1998	2	BO-6	Бостон	Franchise	3615	\$5,091.00
1998	3	BO-1	Бостон	Company	3936	\$6,307.00
1998	3	BO-2	Бостон	Company	3857	\$6,153.00
1998	3	BO-3	Бостон	Company	3898	\$6,319.00
1998	3	BO-4	Бостон	Company	3949	\$6,453.00
1998	3	BO-5	Бостон	Franchise	3617	\$5,052.00
1998	3	BO-6	Бостон	Franchise	3624	\$5,111.00
1998	4	BO-1	Бостон	Company	3853	\$6,021.00
1998	4	BO-2	Бостон	Company	3891	\$6,333.00
1998	4	BO-3	Бостон	Company	3892	\$6,289.00
1998	4	BO-4	Бостон	Company	3966	\$6,571.00
1998	4	BO-5	Бостон	Franchise	3595	\$4,945.00
1998	4	BO-6	Бостон	Franchise	3611	\$5,051.00
1998	1	LA-1	Лос-Анджелес	Franchise	8259	\$29267.00
1998	1	LA-2	Лос-Анджелес	Company	9140	\$31,947.00
1998	1	LA-3	Лос-Анджелес	Company	9727	\$35405.00
1998	1	LA-4	Лос-Анджелес	Franchise	9494	\$33830.00
1998	1	LA-5	Лос-Анджелес	Franchise	10644	\$39,971.00
1998	1	LA-6	Лос-Анджелес	Franchise	10649	\$40,077.00
1998	2	LA-1	Лос-Анджелес	Franchise	9066	\$32,595.00
1998	2	LA-2	Лос-Анджелес	Company	9789	\$35,217.00
1998	2	LA-3	Лос-Анджелес	Company	9814	\$35,455.00
1998	2	LA-4	Лос-Анджелес	Franchise	9917	\$35,926.00
1998	2	LA-5	Лос-Анджелес	Franchise	10617	\$39,424.00
1998	2	LA-6	Лос-Анджелес	Franchise	10190	\$38,387.00
1998	3	LA-1	Лос-Анджелес	Franchise	9531	\$33,966.00
1998	3	LA-2	Лос-Анджелес	Company	9698	\$34,419.00
1998	3	LA-3	Лос-Анджелес	Company	9771	\$34,494.00
1998	3	LA-4	Лос-Анджелес	Franchise	10232	\$37,315.00
1998	3	LA-5	Лос-Анджелес	Franchise	10561	\$39,141.00
1998	3	LA-6	Лос-Анджелес	Franchise	10924	\$41,938.00
1998	4	LA-1	Лос-Анджелес	Franchise	9310	\$33,202.00

Год	Период	Код магазина	Город	Тип магазина	Число сделок	Продажи
1998	4	LA-2	Лос-Анджелес	Company	9496	\$33,910.00
1998	4	LA-3	Лос-Анджелес	Company	9596	\$34,500.00
1998	4	LA-4	Лос-Анджелес	Franchise	10050	\$37,274.00
1998	4	LA-5	Лос-Анджелес	Franchise	10440	\$38,304.00
1998	4	LA-6	Лос-Анджелес	Franchise	10778	\$40,965.00
1998	1	NY-1	Нью-Йорк	Company	6390	\$19,890.00
1998	1	NY-2	Нью-Йорк	Franchise	7016	\$22,229.00
1998	1	NY-3	Нью-Йорк	Franchise	7293	\$24,077.00
1998	1	NY-4	Нью-Йорк	Company	7037	\$22,704.00
1998	1	NY-5	Нью-Йорк	Franchise	7815	\$26,962.00
1998	1	NY-6	Нью-Йорк	Franchise	6935	\$22,925.00
1998	2	NY-1	Нью-Йорк	Company	6954	\$22,389.00
1998	2	NY-2	Нью-Йорк	Franchise	7531	\$25,324.00
1998	2	NY-3	Нью-Йорк	Franchise	7486	\$24,753.00
1998	2	NY-4	Нью-Йорк	Company	7285	\$24,112.00
1998	2	NY-5	Нью-Йорк	Franchise	7749	\$26,325.00
1998	2	NY-6	Нью-Йорк	Franchise	6881	\$23,123.00
1998	3	NY-1	Нью-Йорк	Company	7256	\$23,330.00
1998	3	NY-2	Нью-Йорк	Franchise	7330	\$24,258.00
1998	3	NY-3	Нью-Йорк	Franchise	7212	\$23,386.00
1998	3	NY-4	Нью-Йорк	Company	7480	\$24,619.00
1998	3	NY-5	Нью-Йорк	Franchise	6771	\$22,189.00
1998	3	NY-6	Нью-Йорк	Franchise	6954	\$23,188.00
1998	4	NY-1	Нью-Йорк	Company	7086	\$22,703.00
1998	4	NY-2	Нью-Йорк	Franchise	7275	\$24,245.00
1998	4	NY-3	Нью-Йорк	Franchise	7121	\$23,025.00
1998	4	NY-4	Нью-Йорк	Company	7562	\$25,329.00
1998	4	NY-5	Нью-Йорк	Franchise	6569	\$20,845.00
1998	4	NY-6	Нью-Йорк	Franchise	6973	\$23,220.00

Учтите, что каждому временному периоду и коду магазина соответствует одна и только одна строка таблицы. В терминах базы данных это значит, что коды магазинов и периоды формируют ключи данных.

На основе данных из табл. 20.1 требуется получить сводную таблицу. Разумеется, сначала нужно определить желаемый способ объединения данных. Ясно, что с помощью сводной таблицы мы узнаем совокупный доход и общее число торговых операций. Остается понять, по каким критериям будут группироваться данные.

Проще всего в этом случае сначала определить самую маленькую группу данных. Для указанной таблицы это будут группы по типу магазина/расположению магазина/временному периоду. Например, может потребоваться узнать общий объем продаж для всех собственных магазинов фирмы в городе Нью-Йорк за период 1.

Дополнительно рассматриваются более крупные группы, например общий объем продаж за год в собственных магазинах фирмы в городе Нью-Йорк, а также все продажи, совершенные в Нью-Йорке.

Напоследок они будут разделены по 1997 и 1998 годам.

Мастер сводных таблиц

Для начала познакомимся с мастером сводных таблиц. Впоследствии вы узнаете, как создать сводную таблицу программным способом.

Первым шагом является выделение исходных данных и выбор команды PivotTable Report (Сводная таблица) в меню Data (Данные). В результате отобразится диалоговое окно, показанное на рис. 20.1. (Рисунки в данной книге иллюстрируют версии Excel 97 и 2000. В Excel XP эти диалоговые окна имеют несколько другой вид).

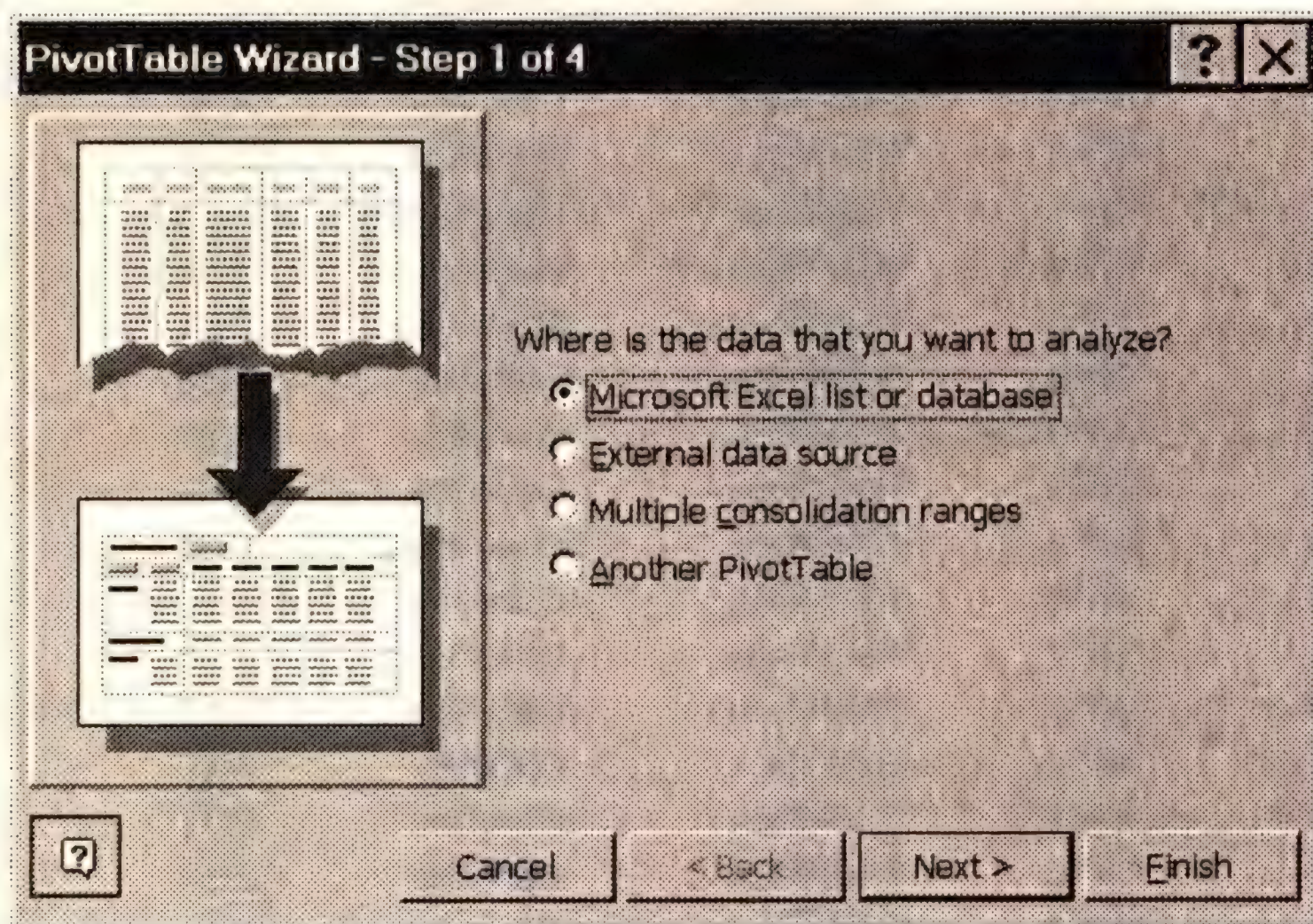


Рис. 20.1. Первое диалоговое окно мастера сводных таблиц

Здесь нужно указать источник данных для сводной таблицы. В данном случае для переключателя Where is the data that you want to analyze? (Создать таблицу на основе данных, находящихся:) следует выбрать позицию Microsoft Excel list or database (В списке или в базе данных Microsoft Excel). Щелкните на кнопке Next (Далее) для перехода к диалоговому окну, показанному на рис. 20.2.

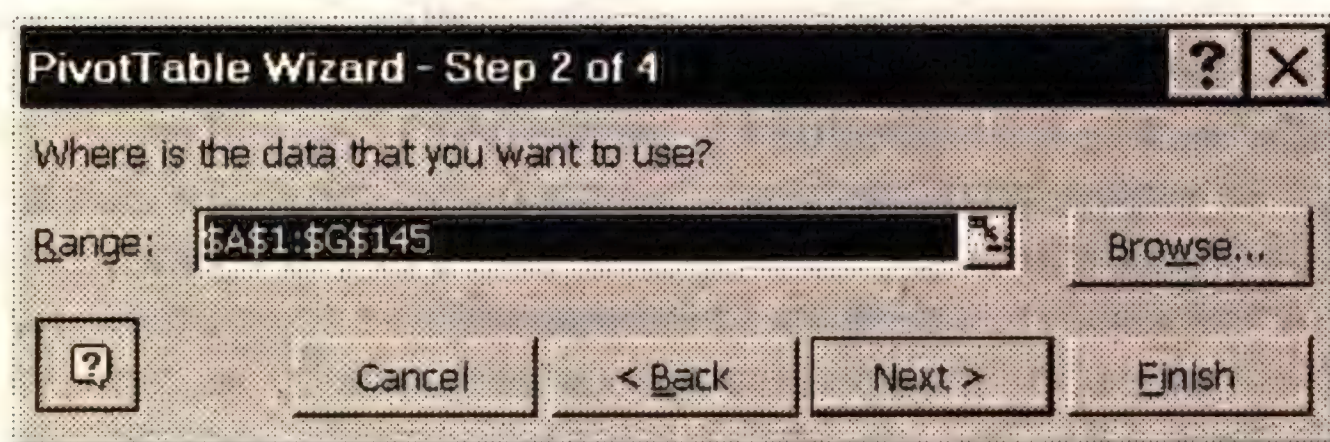


Рис. 20.2. Второе диалоговое окно мастера сводных таблиц

Так как перед запуском мастера был выделен нужный диапазон исходных данных, Excel корректно укажет его в диалоговом окне, как показано на рис. 20.2, поэтому можно просто щелкнуть на кнопке Next (Далее) для перехода к диалоговому окну, показанному на рис. 20.3.

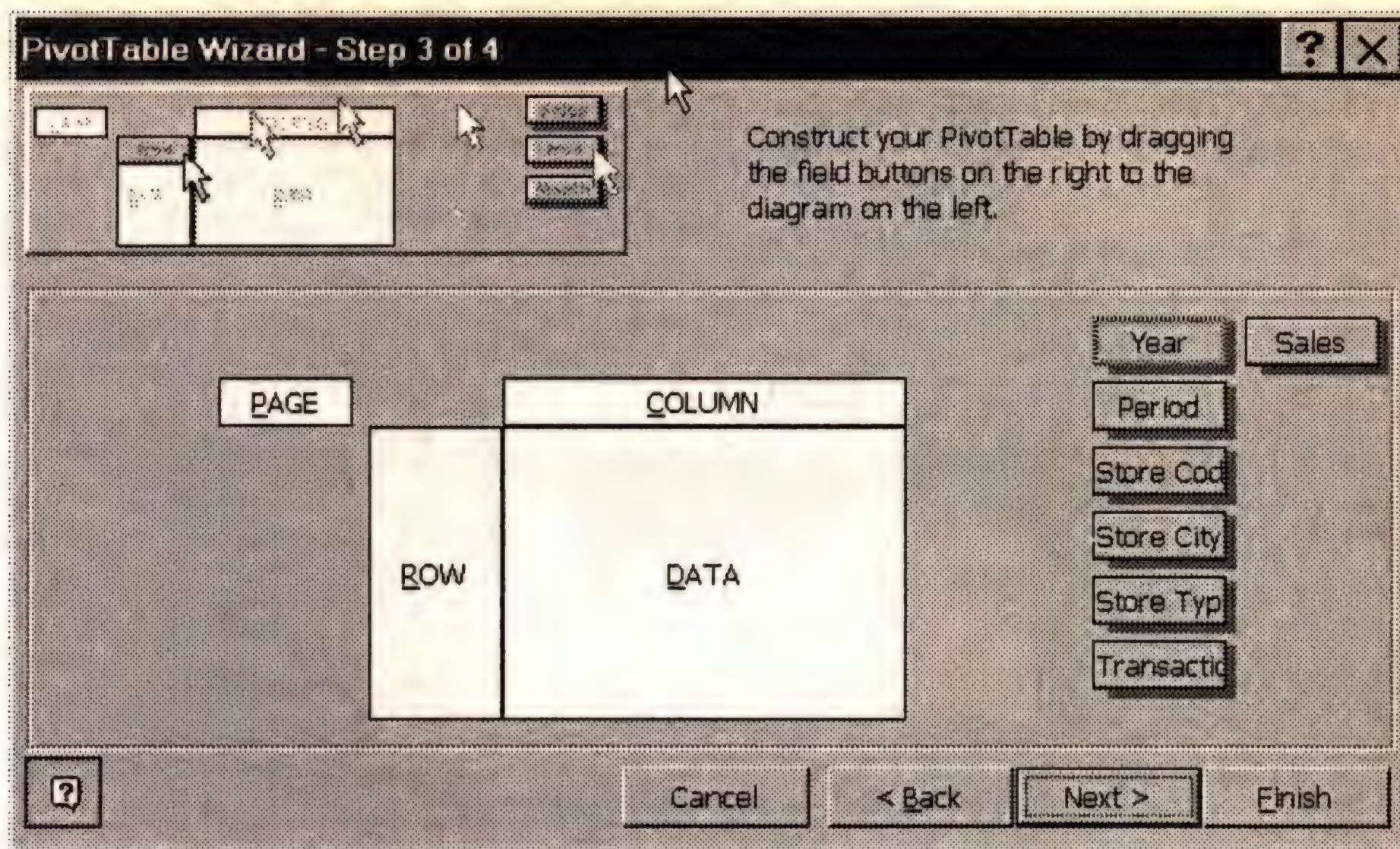


Рис. 20.3. Третье диалоговое окно мастера сводных таблиц

Это диалоговое окно предназначено для форматирования сводной таблицы. Тут требуется указать, какие столбцы исходных данных станут страницами сводной таблицы, какие — строками, какие — столбцами, а какие — данными для объединения. Пользователь должен перетащить расположенные справа кнопки на нужную область. Требуется получить две страницы, по одной для каждого года.

В рассматриваемом примере результат перетаскивания кнопок на различные области показан на рис. 20.4. Обратите внимание на то, что кнопка Store Code никуда не перетащена, так как это поле будет использовано для объединения данных. То есть суммирование производится по кодам магазинов.

Щелчок на кнопке Next (Далее) приведет к появлению диалогового окна, показанного на рис. 20.5, на котором нужно указать местоположение сводной таблицы. Установите переключатель в положение New worksheet (Новый лист) и щелкните на кнопке Finish (Готово).

Появится сводная таблица, отображенная на рис. 20.6.

Обратите внимание: кнопка, управляющая страницами, имеет ярлык Year. Выбор из расположенного справа от нее раскрывающегося списка варианта All, 1997 или 1998 ограничивает показ данных соответствующим периодом. То есть сводная таблица состоит из трех страниц — за 1997 год, за 1998 год и за оба года сразу.

Столбцы соответствуют периодам, а строки — типам магазинов и городам, в которых они расположены. Кроме того, Excel создал дополнительное поле с названием Data, которое используется в качестве подписей к строкам. Оно позволяет приложению понять, что нам требуется суммировать данные. Впрочем, если приложение отреагировало некорректно, исправить ситуацию можно вручную.

Подводя итоги, следует сказать, что основными компонентами сводной таблицы являются страницы, строки, столбцы и поля данных.

Мы не будем продолжать рассказ о работе со сводными таблицами с помощью пользовательского интерфейса Excel, а перейдем к описанию программных методов.

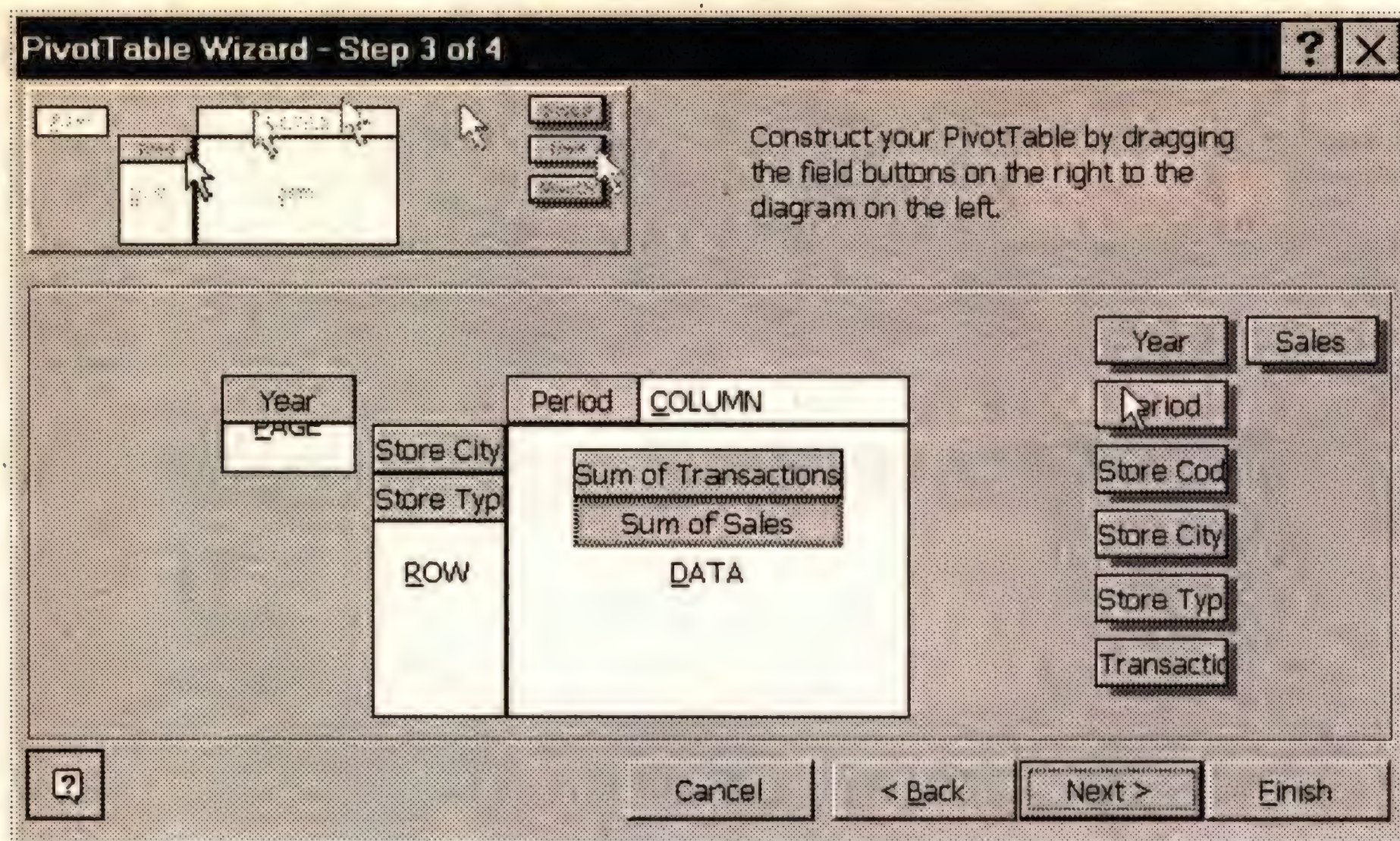


Рис. 20.4. Результат перетаскивания кнопок на различные области



Рис. 20.5. Пятое окно мастера сводных таблиц

Метод PivotTableWizard

Для создания сводной таблицы программным путем используется метод `PivotTableWizard` объекта `Worksheet` или объекта `PivotTable`. Можно предположить, что этот метод открывает диалоговое окно `PivotTable Wizard` (Мастер сводных таблиц), но на самом деле это не так. В применении к объекту `Worksheet` он используется для создания новой сводной таблицы, а в применении к объекту `PivotTable` — для внесения изменений в уже существующую таблицу.

	A	B	C	D	E	F	G	H
1	Year	{All} ▼						
2								
3				Period				
4	Store City	Store Type	Data	1	2	3	4	Grand Total
5	BOSTON	Company	Sum of Transactions	28248	28714	28672	28602	114236
6			Sum of Sales	44678	46927	46256	46223	184084
7		Franchise	Sum of Transactions	13993	13942	13275	13210	54420
8			Sum of Sales	21816	21739	18632	18325	80512
9	BOSTON Sum of Transactions			42241	42656	41947	41812	168656
10	BOSTON Sum of Sales			66494	68666	64888	64548	264596
11	LOS ANGE	Company	Sum of Transactions	34588	35938	35692	35001	141219
12			Sum of Sales	123478	129564	126340	125418	504800
13		Franchise	Sum of Transactions	71583	72947	75619	74392	294541
14			Sum of Sales	262431	268274	279325	274531	1084561
15	LOS ANGELES Sum of Transactions			106171	108885	111311	109393	435760
16	LOS ANGELES Sum of Sales			385909	397838	405665	399949	1589361
17	NEW YOR	Company	Sum of Transactions	24616	26104	27015	26854	104589
18			Sum of Sales	78089	85251	87905	88058	339303
19		Franchise	Sum of Transactions	53273	54351	51822	51218	210664
20			Sum of Sales	176353	182461	170537	167446	696797
21	NEW YORK Sum of Transactions			77889	80455	78837	78072	315253
22	NEW YORK Sum of Sales			254442	267712	258442	255504	1036100
23	Total Sum of Transactions			226301	231996	232095	229277	919669
24	Total Sum of Sales			706845	734216	728995	720001	2890057

Рис. 20.6. Сводная таблица

Вот синтаксис этого метода:

```
expression.PivotTableWizard(SourceType, SourceData, TableDestination, _
    TableName, RowGrand, ColumnGrand, SaveData, HasAutoFormat, _
    AutoPage, Reserved, BackgroundQuery, OptimizeCache, _
    PageFieldOrder, ageFieldWrapCount, ReadData, Connection),
```

где *expression* возвращает объект `Worksheet` или `PivotTable`. Нетрудно догадаться, что параметры метода `PivotTableWizard` соответствуют настройкам диалогового окна `PivotTable Wizard` (Мастер сводных таблиц). С другой стороны, метод `PivotTableWizard` не может быть полностью аналогичен мастеру. Например, с его помощью невозможно указать области строк, столбцов и данных. (Как это сделать, будет рассказано немного позднее). Другими словами, метод `PivotTableWizard` определяет свойства пустой сводной таблицы.

Необязательный параметр *SourceType* указывает на источник данных для сводной таблицы и может принимать значения констант из следующего перечисления:

```
Enum XlPivotTableSourceType
    xlPivotTable = -4148
    xlDatabase = 1
    xlExternal = 2
    xlConsolidation = 3
End Enum
```

Эти значения непосредственно совпадают с положениями переключателя в первом диалоговом окне мастера сводных таблиц, показанного на рис. 20.1.

Если определено значение параметра *SourceType*, требуется указать также значение параметра *SourceData*. Если ни один из этих параметров не указан, приложение использует тип источника данных *xlDatabase*, а источником данных служит именованный диапазон *Database*. При его отсутствии Excel использует текущий выделенный диапазон, если он представляет собой более десяти ячеек с данными. В противном случае метод не будет работать. В конечном счете пользователю проще указать эти параметры, чтобы точно знать, что происходит.

Параметр *SourceData* указывает источник данных для сводной таблицы. Это может быть объект *Range*, набор массивов или даже текстовая константа, представляющая собой имя другой сводной таблицы. Для внешних данных это должен быть двухэлементный массив, первый элемент которого является связывающей строкой, указывающей на источник данных во внешней базе, а второй элемент — строкой запроса SQL, используемой для получения данных.

Параметр *TableDestination* представляет собой объект *Range*, определяющий, куда следует поместить сводную таблицу. Он может включать указание на конкретный лист, на котором будет располагаться таблица.

Параметр *TableName* является строкой, определяющей имя новой сводной таблицы.

Чтобы сделать видимой общую сумму данных в строках сводной таблицы, параметру *RowGrand* следует присвоить значение *True*. Соответственно, присвоение параметру *ColumnGrand* значения *True* делает видимой общую сумму данных в столбцах.

Для сохранения данных сводной таблицы параметру *SaveData* нужно присвоить значение *True*. Если этот параметр имеет значение *False*, сохраняется только определение таблицы.

Присвоение значения *True* параметру *HasAutoFormat* обеспечивает автоматическое форматирование сводной таблицы при обновлении и перемещении любых полей.

Параметры *PageFieldOrder* и *PageFieldWrapCount* имеют значение только при наличии нескольких страниц. Они указывают, как именно относительно друг друга располагаются кнопки полей, соответствующих страницам, и раскрывающиеся списки. Параметр *PageFieldOrder* может принимать значение *xlDownThenOver* (по умолчанию) или *xlOverThenDown*. Например, при наличии трех страниц запись:

```
PageFieldOrder = xlDownThenOver
PageFieldWrapCount = 2
```

расположит поля страниц, как показано на рис. 20.7.

	A	B	C	D	E	F
1						
2	Year	(All) ▼		Store Type	(All) ▼	
3	Store City	(All) ▼				
4						
5		Period				
6	Data	1	2	3	4	Grand Total
7	Sum of Transactions	226301	231996	232095	229277	919669
8	Sum of Sales	706845	734216	728995	720001	2890057
9						

Рис. 20.7. Иллюстрация порядка расположения полей страниц

Эта сводная таблица предназначена всего лишь для иллюстрации положения полей страниц. Она получена из исходной таблицы путем перемещения полей строк на поля страниц. Присвоение параметру PageFieldOrder значения xlOverThenDown приведет к взаимной замене полей Store City и Store Type.

Следующий код просто создает сводную таблицу, показанную на рис. 20.6, в месте расположения активной ячейки:

```
ActiveSheet.PivotTableWizard _  
    SourceType:=xlDatabase.  
    SourceData:="'Source'!R1C1:R145C7".  
    TableName:="Sales&Trans"
```

Результат выполнения данного кода показан на рис. 20.8. Причина, по которой возникает впечатление, что ничего не произошло, уже упоминалась ранее. Дело в том, что метод PivotTableWizard не позволяет указать, какие поля соответствуют страницам, столбцам, строкам и данным. Именно поэтому на рис. 20.8 показана пустая сводная таблица.

	A	B
1		Total
2	Total	

Рис. 20.8. Пустая сводная таблица

Объект PivotTable

Чтобы лучше представить порядок следующих действий, рассмотрим объект PivotTable и различные семейства его потомков.

С помощью метода PivotTableWizard была создана сводная таблица Sales&Trans. Все объекты PivotTable имеют семейство PivotFields, доступ к которому осуществляется с помощью свойства PivotFields. Соответственно, код:

```
Dim pf As PivotField  
For Each pf In _  
    ActiveSheet.PivotTables("Sales&Trans").PivotFields  
    Debug.Print pf.Name  
Next
```

сформирует следующий список полей сводной таблицы:

- Year
- Period
- Store Code
- Store City
- Store Type
- Transactions
- Sales

Теперь каждый объект PivotField может иметь обозначение, которое указывает, должно ли поле использоваться в качестве области строки, области столбца, области страницы или области данных.

Разместить поле сводной таблицы в определенной области можно различными способами. Один подход использует свойство `Orientation` полей сводной таблицы, другой — метод `AddFields`. К сожалению, в документации не содержится исчерпывающей информации ни по одному из этих подходов, поэтому их изучение происходит в основном путем экспериментов.

Чтобы понять принцип использования свойства `Orientation`, рассмотрим код, показанный в листинге 20.1. Он задает одновременно свойства `Orientation` и `Position`. Особенности этого кода будут обсуждаться немного позже.

Листинг 20.1. Процедура `CreatePivotFields`

```
Sub CreatePivotFields()

    ' Источником данных для сводной таблицы
    ' является лист с именем 'Source'

    ActiveSheet.PivotTableWizard _
        SourceType:=xlDatabase, _
        SourceData:="'Source'!R1C1:R145C7", _
        TableName:="PivotTable1"

    With ActiveSheet.PivotTables("Sales&Trans")

        Debug.Print "Before all:"
        ShowFields

        .PivotFields("Year").Orientation = xlPageField
        .PivotFields("Year").Position = 1

        .PivotFields("Store City").Orientation = xlRowField
        .PivotFields("Store City").Position = 1

        .PivotFields("Store Type").Orientation = xlRowField
        .PivotFields("Store Type").Position = 2

        .PivotFields("Period").Orientation = xlColumnField
        Debug.Print "Before data fields:"
        ShowFields

        With .PivotFields("Transactions")
            .Orientation = xlDataField
            .Position = 1
        End With
        With .PivotFields("Sales")
            .Orientation = xlDataField
            .Position = 2
        End With
        Debug.Print ""
        Debug.Print "After data fields:"
        ShowFields
        .PivotFields("Data").Orientation = xlRowField
        .PivotFields("Data").Position = 3
    End With
End Sub
```

Код процедуры `ShowFields`, вызываемой внутри процедуры `CreatePivotFields`, показан в листинге 20.2. `ShowFields` выводит в окне `Immediate` список всех полей сводной таблицы, что весьма полезно при экспериментах и отладке.

Листинг 20.2. Процедура ShowFields

```
Sub ShowFields()  
Dim pf As PivotField  
Debug.Print "*PivotFields:"  
For Each pf In  
    ActiveSheet.PivotTables("Sales&Trans").PivotFields  
        Debug.Print pf.Name  
Next  
Debug.Print "*RowFields:"  
For Each pf In  
    ActiveSheet.PivotTables("Sales&Trans").RowFields  
        Debug.Print pf.Name  
Next  
Debug.Print "*ColFields:"  
For Each pf In  
    ActiveSheet.PivotTables("Sales&Trans").ColumnFields  
        Debug.Print pf.Name  
Next  
Debug.Print "*DataFields:"  
For Each pf In  
    ActiveSheet.PivotTables("Sales&Trans").DataFields  
        Debug.Print pf.Name  
Next  
End Sub
```

Запуск процедуры CreatePivotFields приводит к появлению в окне Immediate следующего списка:

Сначала:

- *PivotFields:
- Year
- Period
- Store Code
- Store City
- Store Type
- Transactions
- Sales
- *RowFields:
- *ColFields:
- *DataFields:

Перед полями данных:

- *PivotFields:
- Year
- Period
- Store Code
- Store City
- Store Type
- Transactions
- Sales

- *RowFields:

- Store City

- Store Type

- *ColFields:

- Period

- *DataFields:

После полей данных:

- *PivotFields:

- Year

- Period

- Store Code

- Store City

- Store Type

- Transactions

- Sales

- Data

- *RowFields:

- Store City

- Store Type

- Data

- *ColFields:

- Period

- *DataFields:

- Sum of Transactions

- Sum of Sales

Данный список определяет, что специальное поле с именем Data создается приложением только при значении xlDataField полей Transactions и Sales. Это утверждение поддерживается и тем фактом, что если поместить две последние строки кода:

```
.PivotFields("Data").Orientation = xlRowField  
.PivotFields("Data").Position = 3
```

непосредственно перед оператором With, относящимся к полю Transactions, при попытке запуска процедуры появится сообщение об ошибке. Выполнение кода будет остановлено на строке:

```
.PivotFields("Data").Orientation = xlRowField.
```

поскольку невозможно определить свойство Orientation несуществующего поля Data.

Теперь осталось убедиться в том, что для областей строк, столбцов и страниц приложение просто добавляет поля сводной таблицы в соответствующее семейство. При этом для области данных создаются новые поля с именами Sum of Transactions и Sum of Sales, которые считаются полями данных, но не относятся к полям сводной таблицы!

Именованные поля данных

Следует сделать несколько замечаний по поводу именованных полей данных. Важно понимать, что если имя поля данных не было изменено, но пользователь заменил обобщенную функцию, например, с суммирования на усреднение, приложение автоматически переименует поле данных. В нашем случае это будет замена названия с Sum of Sales на Average of Sales. Однако изменения вручную присвоенных полям данных имен при замене обобщенной функции не происходит.

Для переименования поля данных достаточно присвоить новое значение его свойству Name. Однако, несмотря на то что поле Data не принадлежит к семейству PivotFields, использовать имя поля сводной таблицы в качестве имени поля данных нельзя. Например, невозможно переименовать Sum of Transactions в Transactions, так как это имя уже присвоено полю сводной таблицы. (Попытка сделать это приведет к сообщению об ошибке.) Соответственно, при составлении таблицы данных, на основе которой будет сформирована сводная таблица, выбор заголовков столбцов должен производиться с учетом изложенных выше соображений!

Итоговая программа

Итак, пришло время собрать в одном месте код, который требуется для создания показанной на рис. 20.6 сводной таблицы. Он показан в листинге 20.3.

Листинг 20.3. Процедура CreatePivot

```
Sub CreatePivot()
```

```
' Создание сводной таблицы в активной ячейке  
' Таблица с данными находится на листе с именем Source
```

```
ActiveSheet.PivotTableWizard _  
    SourceType:=xlDatabase, _  
    SourceData:="'Company Both'!R1C1:R145C7", _  
    TableName:="Sales&Trans"
```

```
' Укажем, в каких областях сводной таблицы должны располагаться ее поля  
With ActiveSheet.PivotTables("Sales&Trans")
```

```
    .PivotFields("Year").Orientation = xlPageField  
    .PivotFields("Year").Position = 1
```

```
    .PivotFields("Store City").Orientation = _  
        xlRowField  
    .PivotFields("Store City").Position = 1
```

```
    .PivotFields("Store Type").Orientation = _  
        xlRowField  
    .PivotFields("Store Type").Position = 2
```

```
    .PivotFields("Period").Orientation = _  
        xlColumnField
```

```
With .PivotFields("Transactions")  
    .Orientation = xlDataField  
    .Position = 1
```

```
End With
```

```
With .PivotFields("Sales")  
    .Orientation = xlDataField  
    .Position = 2
```



```

End With
.PivotFields("Data").Orientation = xlRowField
.PivotFields("Data").Position = 3
End With
End Sub

```

Другим способом размещения полей сводной таблицы в ее определенных областях является метод `AddFields` объекта `PivotTable`. К сожалению, его нельзя использовать для полей данных. Вот синтаксис этого метода:

```

PivotTableObject.AddFields(RowFields,
    ColumnFields, PageFields, AddToTable)

```

Необязательный параметр *RowFields* может указывать как одно имя поля сводной таблицы, так и массив таких имен, добавляемый в виде строк. Аналогичен смысл параметров *ColumnFields* и *PageFields*.

Любой вызов метода `AddFields` приводит к замене существующих полей определенного типа (строка, столбец или страница) полями, указанными в параметрах метода. Если требуется добавить, а не заменить поля, параметру `AddToTable` следует присвоить значение `True`.

Альтернатива процедуре `CreatePivot` приводится в листинге 20.4. Для полей строк, столбцов и страниц используется метод `AddFields`. Данная процедура короче предыдущей. Кроме того, именно этот подход использует приложение Excel при записи макросов для создания сводных таблиц.

Листинг 20.4. Создание сводной таблицы с помощью метода `AddFields`

```

Sub CreatePivot2()

' Создание сводной таблицы в активной ячейке
' Таблица с данными находится на листе с именем Source

ActiveSheet.PivotTableWizard _
    SourceType:=xlDatabase, _
    SourceData:="'Source'!R1C1:R145C7", _
    TableName:="Sales&Trans2"

ActiveSheet.PivotTables("Sales&Trans2").AddFields _
    RowFields:=Array("Store City", "Store Type"), _
    ColumnFields:="Period", _
    PageFields:="Year"
With ActiveSheet.PivotTables("Sales&Trans2")

    With .PivotFields("Transactions")
        .Orientation = xlDataField
        .Position = 1
    End With
    With .PivotFields("Sales")
        .Orientation = xlDataField
        .Position = 2
    End With
End With
End Sub

```

Свойства и методы объекта `PivotTable`

Свойства и методы объекта `PivotTable` перечислены в табл. 20.2. Наиболее важные из них будут рассматриваться в связи с выполняемыми ими функциями.

Таблица 20.2. Свойства и методы объекта PivotTable

_Default	ErrorString	PrintTitles<v9>
_PivotSelect<v10>	Format<v9>	RefreshDate
AddDataField<v10>	GetData	RefreshName
AddFields	GetPivotData<v10>	RefreshTable
Application	GrandTotalName<v9>	RepeatItemsOnEachPrintedPage<v9>
CacheIndex	HasAutoFormat	RowFields
CalculatedFields	HiddenFields	RowGrand
CalculatedMembers<v10>	InnerDetail	RowRange
ColumnFields	ListFormulas	SaveData
ColumnGrand	ManualUpdate	SelectionMode
ColumnRange	MDX<v10>	ShowCellBackgroundFromOLAP<v10>
CreateCubeFile<v10>	MergeLabels	ShowPageMultipleItemLabel<v10>
Creator	Name	ShowPages
CubeFields<v9>	NullString	SmallGrid<v9>
DataBodyRange	PageFieldOrder	SourceData
DataFields	PageFields	SubtotalHiddenPageItems
DataLabelRange	PageFieldStyle	TableRange1
DataPivotField<v10>	PageFieldWrapCount	TableRange2
DisplayEmptyColumn<v10>	PageRange	TableStyle
DisplayEmptyRow<v10>	PageRangeCells	Tag
DisplayErrorString	Parent	TotalsAnnotation<v9>
DisplayImmediateItems<v10>	PivotCache	Update
DisplayNullString	PivotFields	VacatedStyle
Dummy15<v10>	PivotFormulas	Value
EnableDataValueEditing<v10>	PivotSelect	Version<v10>
EnableDrilldown	PivotSelection	ViewCalculatedMembers<v10>
EnableFieldDialog	PivotSelectionStandard<v10>	VisibleFields
EnableFieldList<v10>	PivotTableWizard	VisualTotals<v10>
EnableWizard	PreserveFormatting	

Свойства, возвращающие семейство Fields

Некоторые свойства объекта PivotTable предназначены для возвращения семейства полей.

Свойство ColumnFields

Свойство ColumnFields возвращает все поля из области столбцов:

```
PivotTableObject.ColumnFields
```

В качестве альтернативы можно возвращать только поля выделенных столбцов:

```
PivotTableObject.ColumnFields(Index),
```

где параметр Index может являться как одним индексом (номером требуемого поля), так и массивом индексов.

Свойство DataFields

Свойство DataFields возвращает все поля из области данных. Вот его синтаксис:

```
PivotTableObject.DataFields
```

В качестве альтернативы существует возможность вернуть только поля выделенных данных:

```
PivotTableObject.DataFields(Index),
```

где параметр *Index* может являться как одним индексом (номером требуемого поля), так и массивом индексов.

Свойство HiddenFields

Поля сводной таблицы могут быть скрыты. Для этого достаточно присвоить их свойству Orientation значение xlHidden. Свойство HiddenFields возвращает семейство всех скрытых полей:

```
PivotTableObject.HiddenFields
```

В качестве альтернативы существует возможность вернуть только выделенные скрытые поля:

```
PivotTableObject.HiddenFields(Index),
```

где параметр *Index* может являться как одним индексом (номером требуемого поля), так и массивом индексов.

Свойство PageFields

Свойство PageFields возвращает все поля из области страниц:

```
PivotTableObject.PageFields
```

В качестве альтернативы существует возможность вернуть только поля выделенных страниц:

```
PivotTableObject.PageFields(Index),
```

где параметр *Index* может являться как одним индексом (номером требуемого поля), так и массивом индексов.

Свойство PivotFields

Свойство PivotFields возвращает все поля таблицы:

```
PivotTableObject.PivotFields
```

В качестве альтернативы существует возможность вернуть только выделенные поля:

```
PivotTableObject.PivotFields(Index),
```

где параметр *Index* может являться как одним индексом (номером требуемого поля), так и массивом индексов.

Свойство RowFields

Свойство RowFields возвращает все поля из области строк:

```
PivotTableObject.RowFields
```

В качестве альтернативы существует возможность вернуть только поля выделенных строк:

```
PivotTableObject.RowFields(Index),
```


где параметр *Index* может являться как одним индексом (номером требуемого поля), так и массивом индексов.

Свойство VisibleFields

Свойство VisibleFields возвращает набор всех видимых полей таблицы:

```
PivotTableObject.VisibleFields
```

В качестве альтернативы существует возможность вернуть только выделенные видимые поля:

```
PivotTableObject.VisibleFields(Index).
```

где параметр *Index* может являться как одним индексом (номером требуемого поля), так и массивом индексов.

Свойства, влияющие на итоговое значение

Объект PivotTable имеет два свойства, влияющих на показ итоговых значений:

- если свойство ColumnGrand (чтение/запись, тип Boolean) имеет значение True, в таблице отображаются суммы по столбцам;
- если свойство RowGrand (чтение/запись, тип Boolean) имеет значение True, в таблице отображаются суммы по строкам.

Для иллюстрации рассмотрим таблицу, показанную на рис. 20.6. Код:

```
ActiveSheet.PivotTables("Sales&Trans"). _  
    ColumnGrand = False  
ActiveSheet.PivotTables("Sales&Trans"). _  
    RowGrand = False
```

приведет к появлению таблицы, показанной на рис. 20.9, в которой отсутствуют общие суммы.

Также можно запретить отображение итогов по определенному полю, например по полю Store City, как показано на рис. 20.9. Это частное свойство объекта PivotField, поэтому мы поговорим о нем позже, при рассмотрении данного объекта. Пока только упомянем, что отображение итогов по определенному полю управляется свойством Subtotals объекта PivotField. К примеру, следующий код скрывает все итоги, отображаемые в таблице из рис. 20.9:

```
Dim i As Integer  
For i = 1 To 12  
    ActiveSheet.PivotTables("Sales&Trans").  
        PivotFields("Store City").Subtotals(i) = False  
Next
```

(В этой таблице 12 различных итогов, причем требуется убрать их все.) В результате получаем таблицу, показанную на рис. 20.10.

Возвращение части сводной таблицы

Некоторые свойства объекта PivotTable возвращают часть сводной таблицы в виде объекта Range. Поговорим о них чуть подробнее.

Свойство ColumnRange

Свойство ColumnRange возвращает объект Range, содержащий области столбцов сводной таблицы. Вид этого объекта показан на рис. 20.11.

1	Year	(All)					
2							
3				Period			
4	Store City	Store Type	Data	1	2	3	4
5	BOSTON	Company	Sum of Transactions	28248	28714	28672	28602
6			Sum of Sales	44678	46927	46256	46223
7		Franchise	Sum of Transactions	13993	13942	13275	13210
8			Sum of Sales	21816	21739	18632	18325
9	BOSTON Sum of Transactions			42241	42656	41947	41812
10	BOSTON Sum of Sales			66494	68666	64888	64548
11	LOS ANGE	Company	Sum of Transactions	34588	35938	35692	35001
12			Sum of Sales	123478	129564	126340	125418
13		Franchise	Sum of Transactions	71583	72947	75619	74392
14			Sum of Sales	262431	268274	279325	274531
15	LOS ANGELES Sum of Transactions			106171	108885	111311	109393
16	LOS ANGELES Sum of Sales			385909	397838	405665	399949
17	NEW YOR	Company	Sum of Transactions	24616	26104	27015	26854
18			Sum of Sales	78089	85251	87905	88058
19		Franchise	Sum of Transactions	53273	54351	51822	51218
20			Sum of Sales	176353	182461	170537	167446
21	NEW YORK Sum of Transactions			77889	80455	78837	78072
22	NEW YORK Sum of Sales			254442	267712	258442	255504

Рис. 20.9. Итоговые суммы не отображаются

	A	B	C	D	E	F	G
1	Year	(All)					
2							
3				Period			
4	Store City	Store Type	Data	1	2	3	4
5	BOSTON	Company	Sum of Transactions	28248	28714	28672	28602
6			Sum of Sales	44678	46927	46256	46223
7		Franchise	Sum of Transactions	13993	13942	13275	13210
8			Sum of Sales	21816	21739	18632	18325
9	LOS ANGE	Company	Sum of Transactions	34588	35938	35692	35001
10			Sum of Sales	123478	129564	126340	125418
11		Franchise	Sum of Transactions	71583	72947	75619	74392
12			Sum of Sales	262431	268274	279325	274531
13	NEW YOR	Company	Sum of Transactions	24616	26104	27015	26854
14			Sum of Sales	78089	85251	87905	88058
15		Franchise	Sum of Transactions	53273	54351	51822	51218
16			Sum of Sales	176353	182461	170537	167446

Рис. 20.10. Итоги отсутствуют вообще

	A	B	C	D	E	F	G
1	Year	(All) ▼					
2							
3				Period			
4	Store City	Store Type	Data	1	2	3	4
5	BOSTON	Company	Sum of Transactions	28248	28714	28672	28602
6			Sum of Sales	44678	46927	46256	46223
7		Franchise	Sum of Transactions	13993	13942	13275	13210
8			Sum of Sales	21816	21739	18632	18325
9	LOS ANGE	Company	Sum of Transactions	34588	35938	35692	35001
10			Sum of Sales	123478	129564	126340	125418
11		Franchise	Sum of Transactions	71583	72947	75619	74392
12			Sum of Sales	262431	268274	279325	274531
13	NEW YOR	Company	Sum of Transactions	24616	26104	27015	26854
14			Sum of Sales	78089	85251	87905	88058
15		Franchise	Sum of Transactions	53273	54351	51822	51218
16			Sum of Sales	176353	182461	170537	167446

Рис. 20.11. Диапазон ColumnRange

Свойство DataBodyRange

Свойство DataBodyRange возвращает объект Range, содержащий область данных сводной таблицы. Результат применения этого свойства показан на рис. 20.12.

	A	B	C	D	E	F	G
1	Year	(All) ▼					
2							
3				Period			
4	Store City	Store Type	Data	1	2	3	4
5	BOSTON	Company	Sum of Transactions	28248	28714	28672	28602
6			Sum of Sales	44678	46927	46256	46223
7		Franchise	Sum of Transactions	13993	13942	13275	13210
8			Sum of Sales	21816	21739	18632	18325
9	LOS ANGE	Company	Sum of Transactions	34588	35938	35692	35001
10			Sum of Sales	123478	129564	126340	125418
11		Franchise	Sum of Transactions	71583	72947	75619	74392
12			Sum of Sales	262431	268274	279325	274531
13	NEW YOR	Company	Sum of Transactions	24616	26104	27015	26854
14			Sum of Sales	78089	85251	87905	88058
15		Franchise	Sum of Transactions	53273	54351	51822	51218
16			Sum of Sales	176353	182461	170537	167446

Рис. 20.12. Диапазон DataBodyRange

Свойство DataLabelRange

Предназначенное только для чтения свойство DataLabelRange возвращает объект Range, который содержит названия полей в области данных сводной таблицы. Результат применения этого свойства показан на рис. 20.13.

	A	B	C	D	E	F	G
1	Year	(All) ▼					
2							
3				Period			
4	Store City	Store Type	Data	1	2	3	4
5	BOSTON	Company	Sum of Transactions	28248	28714	28672	28602
6			Sum of Sales	44678	46927	46256	46223
7		Franchise	Sum of Transactions	13993	13942	13275	13210
8			Sum of Sales	21816	21739	18632	18325
9	LOS ANGE	Company	Sum of Transactions	34588	35938	35692	35001
10			Sum of Sales	123478	129564	126340	125418
11		Franchise	Sum of Transactions	71583	72947	75619	74392
12			Sum of Sales	262431	268274	279325	274531
13	NEW YOR	Company	Sum of Transactions	24616	26104	27015	26854
14			Sum of Sales	78089	85251	87905	88058
15		Franchise	Sum of Transactions	53273	54351	51822	51218
16			Sum of Sales	176353	182461	170537	167446

Рис. 20.13. Диапазон DataLabelRange

Свойства PageRange и PageRangeCells

Свойство PageRange возвращает объект Range, содержащий область страниц сводной таблицы. Это наименьшая прямоугольная область, которая включает все ячейки, связанные с областью страниц.

Свойство PageRangeCells возвращает объект Range, включающий только те ячейки сводной таблицы, которые содержат кнопки области страницы и раскрывающиеся списки. Разница между этими свойствами проиллюстрирована на рис. 20.14 и 20.15.

Свойство RowRange

Свойство RowRange возвращает объект Range, содержащий область строк. Результат применения этого свойства показан на рис. 20.16.

Свойство TableRange1

Свойство TableRange1 возвращает объект Range, содержащий область строк, столбцов и данных, но не область страниц. Результат применения этого свойства показан на рис. 20.17.

Свойство TableRange2

Свойство TableRange2 возвращает объект Range, представляющий собой всю сводную таблицу, то есть область строк, столбцов, данных и страниц. Результат применения этого свойства показан на рис. 20.18.

1						
2	Year	(All)		Store Type	(All)	
3	Store City	(All)				
4						
5		Period				
6	Data	1	2	3	4	Grand Total
7	Sum of Transactions	226301	231996	232095	229277	919669
8	Sum of Sales	706845	734216	728995	720001	2890057

Рис. 20.14. Диапазон PageRange

1						
2	Year	(All)		Store Type	(All)	
3	Store City	(All)				
4						
5		Period				
6	Data	1	2	3	4	Grand Total
7	Sum of Transactions	226301	231996	232095	229277	919669
8	Sum of Sales	706845	734216	728995	720001	2890057

Рис. 20.15. Диапазон PageRangeCells

	A	B	C	D	E	F	G
1	Year	(All)					
2							
3				Period			
4	Store City	Store Type	Data	1	2	3	4
5	BOSTON	Company	Sum of Transactions	28248	28714	28672	28602
6			Sum of Sales	44678	46927	46256	46223
7		Franchise	Sum of Transactions	13993	13942	13275	13210
8			Sum of Sales	21816	21739	18632	18325
9	LOS ANGE	Company	Sum of Transactions	34588	35938	35692	35001
10			Sum of Sales	123478	129564	126340	125418
11		Franchise	Sum of Transactions	71583	72947	75619	74392
12			Sum of Sales	262431	268274	279325	274531
13	NEW YOR	Company	Sum of Transactions	24616	26104	27015	26854
14			Sum of Sales	78089	85251	87905	88058
15		Franchise	Sum of Transactions	53273	54351	51822	51218
16			Sum of Sales	176353	182461	170537	167446

Рис. 20.16. Диапазон RowRange

Метод PivotSelect и свойство PivotSelection

Метод PivotSelect выделяет часть сводной таблицы:

```
PivotTableObject.PivotSelect(Name, Mode)
```


	A	B	C	D	E	F	G
1	Year	(All) ▼					
2							
3				Period			
4	Store City	Store Type	Data	1	2	3	4
5	BOSTON	Company	Sum of Transactions	28248	28714	28672	28602
6			Sum of Sales	44678	46927	46256	46223
7		Franchise	Sum of Transactions	13993	13942	13275	13210
8			Sum of Sales	21816	21739	18632	18325
9	LOS ANGE	Company	Sum of Transactions	34588	35938	35692	35001
10			Sum of Sales	123478	129564	126340	125418
11		Franchise	Sum of Transactions	71583	72947	75619	74392
12			Sum of Sales	262431	268274	279325	274531
13	NEW YOR	Company	Sum of Transactions	24616	26104	27015	26854
14			Sum of Sales	78089	85251	87905	88058
15		Franchise	Sum of Transactions	53273	54351	51822	51218
16			Sum of Sales	176353	182461	170537	167446

Рис. 20.17. Диапазон TableRange1

	A	B	C	D	E	F	G
1	Year	(All) ▼					
2							
3				Period			
4	Store City	Store Type	Data	1	2	3	4
5	BOSTON	Company	Sum of Transactions	28248	28714	28672	28602
6			Sum of Sales	44678	46927	46256	46223
7		Franchise	Sum of Transactions	13993	13942	13275	13210
8			Sum of Sales	21816	21739	18632	18325
9	LOS ANGE	Company	Sum of Transactions	34588	35938	35692	35001
10			Sum of Sales	123478	129564	126340	125418
11		Franchise	Sum of Transactions	71583	72947	75619	74392
12			Sum of Sales	262431	268274	279325	274531
13	NEW YOR	Company	Sum of Transactions	24616	26104	27015	26854
14			Sum of Sales	78089	85251	87905	88058
15		Franchise	Sum of Transactions	53273	54351	51822	51218
16			Sum of Sales	176353	182461	170537	167446

Рис. 20.18. Диапазон TableRange2

Параметр *Mode* определяет режим выделения и может принимать значения констант из следующего перечисления:

```
Enum X1PTSelectionMode
    xlDataAndLabel = 0
    xlLabelOnly = 1
    xlDataOnly = 2
```



```

xlOrigin = 3
xlBlanks = 4
xlButton = 15
xlFirstRow = 256      ' Только для версии Excel 9
End Enum

```

Параметр *Name* указывает на выделенный набор, на который в документации Microsoft ссылаются как на «стандартный формат выделения сводной таблицы». К сожалению, там не объясняется, что же это значит. Вместо этого отмечено следующее: «Строковое выражение, используемое для выделения части сводной таблицы. Проще всего понять его синтаксис, включив запись макросов, выделив ячейки сводной таблицы и внимательно рассмотрев полученный в результате код». Дополнительные сведения можно найти в справочной документации Excel VBA (начните поиск с темы PivotSelect).

Рассмотрим ряд примеров, основанных на сводной таблице из рис. 20.10. Однако для иллюстрации свойства Name и для уменьшения размера рисунков, переименуем поле Sum of Transactions в Trans, а поле Sum of Sales в Sale с помощью следующего кода:

```

Sub Rename()

' Для сокращения имен в полях данных

ActiveSheet.PivotTables("Sales&Trans").
    DataFields("Sum of Transactions").Name = "Trans"

ActiveSheet.PivotTables("Sales&Trans").
    DataFields("Sum of Sales").Name = "Sale"

End Sub

```

Это еще раз подтверждает сказанное ранее, а именно: несмотря на наше желание переименовать поле Sum of Sales в Sales, сделать это невозможно, потому что уже существует столбец исходной таблицы с таким именем. Поэтому для переименования было выбрано имя Sale. А теперь вернемся к коду.

Вот код, выделяющий всю сводную таблицу:

```

ActiveSheet.PivotTables("Sales&Trans").
    PivotSelect "", xlDataAndLabel

```

Ниже приводится код, выделяющий область меток с названиями поля Store Type (название поля сводной таблицы и элементов раскрывающегося списка):

```

ActiveSheet.PivotTables("Sales&Trans").
    PivotSelect "'Store Type'[All]", xlLabelOnly

```

Следующий код выделяет все данные и метки, относящиеся к элементу Company:

```

ActiveSheet.PivotTables("Sales&Trans").
    PivotSelect "Company", xlDataAndLabel

```

Далее следует код, выделяющий ячейки, показанные на рис. 20.19:

```

ActiveSheet.PivotTables("Sales&Trans").
    PivotSelect "Company BOSTON", xlDataAndLabel

```

С другой стороны, поменяв местами слова Company и BOSTON:

```

ActiveSheet.PivotTables("Sales&Trans").
    PivotSelect "BOSTON Company", xlDataAndLabel

```

получим выделенный набор, показанный на рис. 20.20, в который не входит метка Company!

	A	B	C	D	E	F	G
1	Year	(All) ▼					
2							
3				Period			
4	Store City	Store Type	Data	1	2	3	4
5	BOSTON	Company	Trans	28248	28714	28672	28602
6			Sale	44678	46927	46256	46223
7		Franchise	Trans	13993	13942	13275	13210
8			Sale	21816	21739	18632	18325
9	LOS ANGELES	Company	Trans	34588	35938	35692	35001
10			Sale	123478	129564	126340	125418
11		Franchise	Trans	71583	72947	75619	74392
12			Sale	262431	268274	279325	274531
13	NEW YORK	Company	Trans	24616	26104	27015	26854
14			Sale	78089	85251	87905	88058
15		Franchise	Trans	53273	54351	51822	51218
16			Sale	176353	182461	170537	167446

Рис. 20.19. Выделение метки типа магазина и данных для города Бостон

	A	B	C	D	E	F	G
1	Year	(All) ▼					
2							
3				Period			
4	Store City	Store Type	Data	1	2	3	4
5	BOSTON	Company	Trans	28248	28714	28672	28602
6			Sale	44678	46927	46256	46223
7		Franchise	Trans	13993	13942	13275	13210
8			Sale	21816	21739	18632	18325
9	LOS ANGELES	Company	Trans	34588	35938	35692	35001
10			Sale	123478	129564	126340	125418
11		Franchise	Trans	71583	72947	75619	74392
12			Sale	262431	268274	279325	274531
13	NEW YORK	Company	Trans	24616	26104	27015	26854
14			Sale	78089	85251	87905	88058
15		Franchise	Trans	53273	54351	51822	51218
16			Sale	176353	182461	170537	167446

Рис. 20.20. В результате изменения порядка слов были выбраны данные для собственных магазинов фирмы исключительно в городе Бостон

Следующий код выделяет ячейку E12 сводной таблицы с рис. 20.10:

```
ActiveSheet.PivotTables("Sales&Trans").PivotSelect _
    "'LOS ANGELES' Franchise 'Sale' '2'", xlDataOnly
```

Вот код, выделяющий метки и данные для Бостона и Нью-Йорка:

```
ActiveSheet.PivotTables("Sales&Trans"). _
    PivotSelect "'Store City'[BOSTON,'NEW YORK']", xlDataAndLabel
```


Если заменить запятую двоеточием:

```
ActiveSheet.PivotTables("Sales&Trans").  
PivotSelect "'Store City'[BOSTON:'NEW YORK']". xlDataAndLabel.
```

будут выделены все элементы, начиная с Бостона и заканчивая Нью-Йорком.

Свойство PivotSelection возвращает или задает выделенные элементы сводной таблицы опять же в стандартном формате выделения. Определение этого свойства эквивалентно вызову метода PivotSelect со значением аргумента Mode, равным xlDataAndLabel.

Дополнительные свойства и методы объекта PivotTable

Рассмотрим несколько дополнительных свойств и методов объекта PivotTable.

Метод AddFields

Этот метод вы уже наблюдали в действии ранее.

Метод CalculatedFields

В сводную таблицу можно добавить вычисляемые поля, то есть поля, которые не являются частью исходной таблицы с данными, а вычисляются на основе этих данных по некоей формуле.

Метод CalculatedFields возвращает семейство CalculatedFields, включающее в себя все вычисляемые поля. Для добавления нового поля используется метод Add из семейства CalculatedFields:

```
CalculatedFieldsObject.Add(Name, Formula).
```

где параметр *Name* представляет собой имя поля, а параметр *Formula* — формулу, по которой производятся вычисления.

Для иллюстрации рассмотрим код, создающий вычисляемое поле в таблице с рис. 20.10. Результат вычислений показан на рис. 20.21.

```
With ActiveSheet.PivotTables("Sales&Trans").  
CalculatedFields.Add("Average", _  
    "= Sales/Transactions")  
    .Orientation = xlDataField  
    .Name = "Avg Check"  
    .NumberFormat = "##.##"  
End With
```

Значения вычисляемых полей подсчитываются непосредственно из сводной таблицы. Таблица с исходными данными напрямую не включена в этот процесс. Именно поэтому для вычисляемого поля не указывается обобщенная функция. Ведь она будет просто проигнорирована. Соответственно, к примеру, значение в ячейке D7 получается делением значения ячейки D6 на значение ячейки D5.

Напоследок упомянем о методе ListFormulas, с помощью которого создается список всех вычисляемых полей на отдельном листе.

Свойства, связанные с обработкой ошибок

Когда свойство DisplayErrorString имеет значение True, в ячейках сводной таблицы с ошибками отображается значение свойства ErrorString. По умолчанию свойство DisplayErrorString имеет значение False. Как упоминается в справочной документации Excel, это свойство особенно полезно для устранения ситуации деления на ноль.

	A	B	C	D	E	F	G
1	Year	(All) ▼					
2							
3				Period			
4	Store City	Store Type	Data	1	2	3	4
5	BOSTON	Company	Trans	28248	28714	28672	28602
6			Sale	44678	46927	46256	46223
7			Avg Check	1.6	1.6	1.6	1.6
8		Franchise	Trans	13993	13942	13275	13210
9			Sale	21816	21739	18632	18325
10			Avg Check	1.6	1.6	1.4	1.4
11	LOS ANGE	Company	Trans	34588	35938	35692	35001
12			Sale	123478	129564	126340	125418
13			Avg Check	3.6	3.6	3.5	3.6
14		Franchise	Trans	71583	72947	75619	74392
15			Sale	262431	268274	279325	274531
16			Avg Check	3.7	3.7	3.7	3.7
17	NEW YOR	Company	Trans	24616	26104	27015	26854
18			Sale	78089	85251	87905	88058
19			Avg Check	3.2	3.3	3.3	3.3
20		Franchise	Trans	53273	54351	51822	51218
21			Sale	176353	182461	170537	167446
22			Avg Check	3.3	3.4	3.3	3.3
23							

Рис. 20.21. Результат создания вычисляемого поля

Свойство `ErrorString` задает строку, отображаемую в ячейке с ошибкой, если свойство `DisplayErrorString` имеет значение `True`. Его значением по умолчанию является пустая строка.

Свойства, связанные с нулевыми значениями

Когда свойство `DisplayNullString` имеет заданное по умолчанию значение `True`, в ячейках сводной таблицы, которые содержат нулевые значения, будет отображаться значение свойства `NullString`.

Свойство `NullString` задает строку, отображающуюся в ячейках с нулевыми значениями. Его значением по умолчанию является пустая строка.

Свойство `EnableDrillDown`

Сводные таблицы допускают перемещение по иерархии объектов. Для иллюстрации дважды щелкнем на ячейке D5 сводной таблицы с рис. 20.10. Приложение создаст новый лист, вид которого показан на рис. 20.22.

Этот лист отображает строки таблицы с исходными данными, которые вносят вклад в значение ячейки D5. Обратите внимание: строки являются полными, хотя столбец `Sales` не вносит вклада в значение ячейки D5.

К этому моменту вы уже, наверное, догадались, что логическое, допускающее редактирование, свойство `EnableDrillDown` используется для разрешения и запрещения перемещения по иерархии объектов.

	A	B	C	D	E	F	G
1	Year	Period	Store Code	Store City	Store Type	Transactions	Sales
2	1998	1	BO-1	BOSTON	Company	3881	6248
3	1998	1	BO-2	BOSTON	Company	3789	5722
4	1998	1	BO-3	BOSTON	Company	3877	6278
5	1998	1	BO-4	BOSTON	Company	3862	6123
6	1997	1	BO-4	BOSTON	Company	3218	5102
7	1997	1	BO-3	BOSTON	Company	3230	5231
8	1997	1	BO-2	BOSTON	Company	3157	4768
9	1997	1	BO-1	BOSTON	Company	3234	5206

Рис. 20.22. Иллюстрация перемещения по иерархии объектов

Свойства и методы, связанные с форматированием

Когда редактируемое свойство HasAutoFormat имеет значение True (это его значение, заданное по умолчанию), сводная таблица автоматически форматируется при обновлении или при перемещении полей.

Названия полей для строк, столбцов, промежуточных и итоговых сумм на рис. 20.6 не слиты с соседними пустыми ячейками. Однако такое впечатление может возникнуть из-за расположенных рядом границ ячеек. Если требуется осуществить слияние, достаточно присвоить свойству MergeLabels значение True.

Если редактируемое свойство PreserveFormatting имеет заданное по умолчанию значение True, форматирование сводной таблицы сохраняется при обновлении или пересчете с помощью таких операций, как сортировка или изменение элементов области страницы.

Свойство TableStyle возвращает или задает строку с названием стиля сводной таблицы. По умолчанию значением этого свойства является пустая строка, то есть отсутствие стиля.

Обновление сводной таблицы

Если свойство ManualUpdate имеет значение True, пересчет данных в сводной таблице осуществляется только по требованию пользователя. По умолчанию это свойство имеет значение False.

Метод RefreshTable используется для обновления массива данных в кэше сводной таблицы. При успешном обновлении метод возвращает значение True.

Свойство RefreshDate возвращает данные, на основе которых в последний раз производилось обновление сводной таблицы или ее кэша. Это свойство предназначено только для чтения.

Свойство RefreshName возвращает пользовательское имя того, кто последним обновлял данные сводной таблицы или ее кэш.

Свойства, связанные с областью страниц

Как уже упоминалось, свойство PageFieldOrder возвращает или задает порядок добавления полей из области страниц к сводной таблице. Оно может принимать значения констант из перечисления XlOrder: xlDownThenOver или xlOverThenDown. По умолчанию это свойство имеет значение xlDownThenOver.

Напомним также, что свойство `PageFieldWrapCount` возвращает или задает число полей из области страниц в каждом столбце или в каждой строке сводной таблицы.

Свойство `PageFieldStyle` возвращает или задает стиль, используемый в полях области страниц.

Свойство Name

Свойство `Name` возвращает или задает строку с именем сводной таблицы.

Свойство SaveData (чтение/запись, тип Boolean)

Когда свойство `SaveData` имеет заданное по умолчанию значение `True`, данные сводной таблицы сохраняются вместе с книгой. Если же это свойство имеет значение `False`, сохраняется только определение сводной таблицы.

Метод ShowPages

Метод `ShowPages` создает новую сводную таблицу для каждого элемента в указанной области страниц. При этом новые сводные таблицы помещаются на новые листы книги. Вот синтаксис этого метода:

```
PivotTableObject.ShowPages(PageField)
```

Например, применив данный метод к сводной таблице с рис. 20.10:

```
ActiveSheet.PivotTables("Sales&Trans").ShowPages "Year".
```

получим два новых листа. Первый, с именем 1997, будет содержать исходную сводную таблицу, но в поле `Year` останется только значение 1997. Вторым листом будет содержать ту же самую таблицу, но в поле `Year` будет значение 1998. При этом остается возможность менять поле `Year` в любой из этих сводных таблиц. Другими словами, каждая сводная таблица содержит данные для всех значений поля `Year`.

Свойство SourceData

Предназначенное только для чтения свойство `SourceData` возвращает источник данных, на основе которого была построена сводная таблица. Например, если в качестве источника данных использовался один лист Excel, метод `SourceData` вернет строку, описывающую диапазон источника данных. Код:

```
debug.print ActiveSheet.PivotTables("Sales&Trans").SourceData
```

возвращает строку:

```
'Company Both'!R1C1:R145C7
```

Потомки объекта PivotTable

Потомки объекта `PivotTable` показаны на рис. 20.23.

Объект PivotField

Свойства и методы объекта `PivotField` перечислены в табл. 20.3.

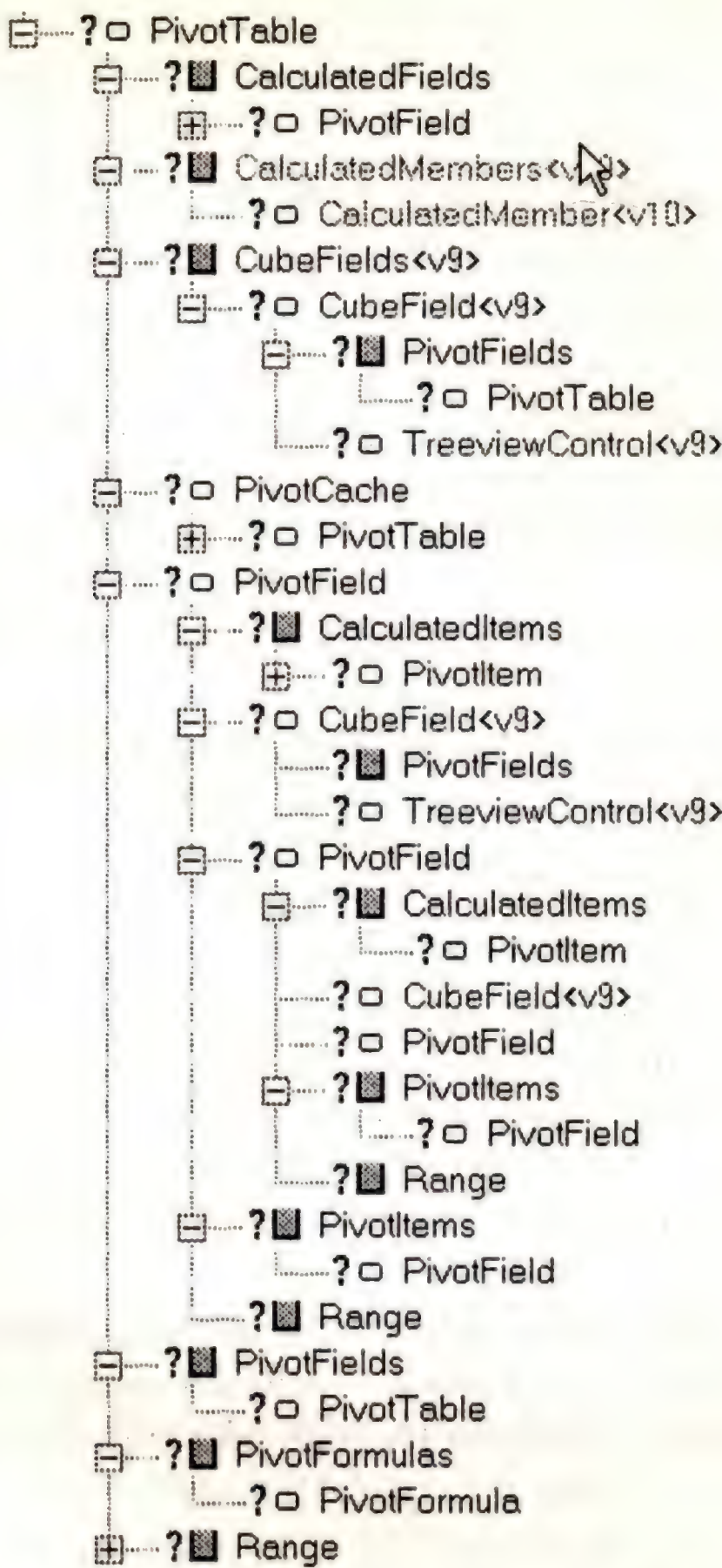


Рис. 20.23. Потомки объекта PivotTable

Таблица 20.3. Свойства и методы объекта PivotField

_Default	CurrentPageName<v9>	LayoutPageBreak<v9>
AddPageItem<v10>	DatabaseSort<v10>	LayoutSubtotalLocation<v9>
Application	DataRange	MemoryUsed
AutoShow	DataType	Name
AutoShowCount	Delete	NumberFormat
AutoShowField	DragToColumn	Orientation
AutoShowRange	DragToData<v9>	Parent
AutoShowType	DragToHide	ParentField
AutoSort	DragToPage	ParentItems
AutoSortField	DragToRow	PivotItems
AutoSortOrder	DrilledDown<v9>	Position
BaseField	EnableItemSelection<v10>	PropertyOrder<v10>
BaseItem	Formula	PropertyParentField<v10>

продолжение ↗

Таблица 20.3 (продолжение)

CalculatedItems	Function	ServerBased
Calculation	GroupLevel	ShowAllItems
Caption<v9>	HiddenItems	SourceName
ChildField	HiddenItemsList<v10>	StandardFormula<v10>
ChildItems	IsCalculated	SubtotalName<v9>
Creator	IsMemberProperty<v10>	Subtotals
CubeField<v9>	LabelRange	TotalLevels
CurrentPage	LayoutBlankLine<v9>	Value
CurrentPageList<v10>	LayoutForm<v9>	VisibleItems

Рассмотрим некоторые из перечисленных в таблице свойств и методов.

Свойства, связанные с методом AutoShow

Метод AutoShow используется для ограничения показа элементов сводной таблицы для определенного поля. Вот его синтаксис:

```
PivotFieldObject.AutoShow(Type, Range, Count, Field)
```

Все параметры данного метода являются обязательными.

Параметр *Type* может принимать одно из двух значений: значение xlAutomatic активизирует остальные параметры и, таким образом включает ограничения, а значение xlManual блокирует остальные параметры и заставляет приложение убрать все ограничения, имеющиеся после последнего вызова данного метода со значением параметра *Type*, равным xlAutomatic.

Остальные параметры можно описать следующим образом: они ограничивают элементы сводной таблицы сверху (Range = xlTop) или снизу (Range = xlBottom). Элемент сводной таблицы Count основан на значении в поле Field.

Соответственно, для таблицы с рис. 20.10 код:

```
ActiveSheet.PivotTables("Sales&Trans").  
PivotFields("Store Type").AutoShow  
xlAutomatic, xlTop, 1, "Sale"
```

показывает максимальные результаты по типу магазинов (Range = xlTop, а Count = 1), основанные на значении поля Sale. Результат показан на рис. 20.24.

Нетрудно заметить, что максимальные продажи в Бостоне были в собственных магазинах фирмы, в то время как в других городах максимальный объем продаж присутствовал у магазинов, торгующих по лицензии и под маркой фирмы.

Тот же самый код, что и в предыдущем случае, но с параметром *Type*, имеющим значение xlManual, уберет все ограничения и восстановит исходный вид сводной таблицы:

```
ActiveSheet.PivotTables("Sales&Trans").  
PivotFields("Store Type").AutoShow  
xlManual, xlTop, 1, "Sale"
```

Перечислим свойства, связанные с методом AutoShow:

- предназначенное только для чтения свойство AutoShowCount возвращает количество элементов, автоматически отображаемых в полях сводной таблицы (это параметр Count метода AutoShow);

- предназначенное только для чтения свойство `AutoShowField` возвращает имя поля данных, которое используется для определения показываемых элементов (это параметр `Field` метода `AutoShow`);
- предназначенное только для чтения свойство `AutoShowRange` возвращает значения `xlTop` или `xlBottom` (это значение параметра `Range` метода `AutoShow`);
- предназначенное только для чтения свойство `AutoShowType` возвращает значение `xlAutomatic`, если метод `AutoShow` выполняется для полей сводной таблицы, и значение `xlManual`, если этот метод заблокирован.

	A	B	C	D	E	F	G
1	Year	(All) ▼					
2							
3				Period			
4	Store City	Store Type	Data	1	2	3	4
5	BOSTON	Company	Trans	28248	28714	28672	28602
6			Sale	44678	46927	46256	46223
7	LOS ANGELES	Franchise	Trans	71583	72947	75619	74392
8			Sale	262431	268274	279325	274531
9	NEW YORK	Franchise	Trans	53273	54351	51822	51218
10			Sale	176353	182461	170537	167446

Рис. 20.24. Иллюстрация метода AutoShow

Сортировка полей сводной таблицы

Метод `AutoSort` задает автоматическое правило сортировки полей сводной таблицы:
`PivotFieldObject.AutoSort(Order, Field)`

Параметр `Order` указывает порядок сортировки и может принимать значения констант из следующего перечисления:

```
Enum XlSortOrder
    xlAscending = 1      ' По возрастанию
    xlDescending = 2    ' По убыванию
End Enum
```

Ему также можно присвоить значение `xlManual`, тем самым заблокировав автоматическую сортировку. Параметр `Field` представляет собой имя поля, которое будет использоваться в качестве ключа сортировки.

Например, для таблицы с рис. 20.10 код:

```
ActiveSheet.PivotTables("Sales&Trans"). _
    PivotFields("Store Type").AutoSort _
    xlAscending, "Sale"
```

приведет к сортировке по полю `Sale` и даст в результате таблицу, показанную на рис. 20.25. Обратите внимание на то, как изменился порядок элементов `Store Type` для Бостона по сравнению с другими городами.

Предназначенное только для чтения свойство `AutoSortField` возвращает имя ключевого поля, а свойство `AutoSortOrder` возвращает порядок сортировки элементов (`xlAscending`, `xlDescending` или `xlManual`).

	A	B	C	D	E	F	G
1	Year	(All) ▼					
2							
3				Period			
4	Store City	Store Type	Data	1	2	3	4
5	BOSTON	Franchise	Trans	13993	13942	13275	13210
6			Sale	21816	21739	18632	18325
7		Company	Trans	28248	28714	28672	28602
8			Sale	44678	46927	46256	46223
9	LOS ANGELES	Company	Trans	34588	35938	35692	35001
10			Sale	123478	129564	126340	125418
11		Franchise	Trans	71583	72947	75619	74392
12			Sale	262431	268274	279325	274531
13	NEW YORK	Company	Trans	24616	26104	27015	26854
14			Sale	78089	85251	87905	88058
15		Franchise	Trans	53273	54351	51822	51218
16			Sale	176353	182461	170537	167446

Рис. 20.25. Результат применения метода AutoSort

Основополагающие свойства

Объект PivotField имеет ряд основополагающих свойств, которые практически всегда требуют определения.

Свойство Function

Свойство Function применяется только к полям данных и возвращает или задает обобщенную функцию, используемую для вычисления поля сводной таблицы. Оно может принимать значения констант из следующего перечисления:

```
Enum XlConsolidationFunction
    xlVarP = -4165      ' Несмещенная дисперсия
    xlVar = -4164       ' Смещенная дисперсия
    xlSum = -4157        ' Сумма значений
    xlStDevP = -4156     ' Несмещенное стандартное отклонение
    xlStDev = -4155      ' Смещенное стандартное отклонение
    xlProduct = -4149    ' Произведение значений
    xlMin = -4139        ' Наименьшее значение
    xlMax = -4136        ' Наибольшее значение
    xlCountNums = -4113  ' Количество записей в числовом формате
    xlCount = -4112      ' Количество записей
    xlAverage = -4106    ' Среднее значение
    xlUnknown = 1000     ' Только в Excel 9
End Enum
```

Свойство NumberFormat

Редактируемое свойство NumberFormat содержит строку, определяющую форматирование числа в поле, которое находится в области данных. Если ячейки выделенного диапазона отформатированы другим образом, это свойство возвращает значение Null.

Свойство Orientation

Свойство Orientation применяется для размещения поля сводной таблицы в ее определенной области. Оно может принимать значения констант из следующего перечисления:

```
Enum XlPivotFieldOrientation
    xlHidden = 0      ' Поле не отображается
    xlRowField = 1    ' Область строк
    xlColumnField = 2 ' Область столбцов
    xlPageField = 3   ' Область страниц
    xlDataField = 4   ' Область данных
End Enum
```

Свойство Position

Редактируемое свойство Position возвращает или задает положение поля в определенной области сводной таблицы. Например, если в области страниц размещены два поля сводной таблицы, нумерация начинается снизу. При этом последний раскрывающийся список занимает положение 1.

Выделение диапазонов

Объект PivotField имеет два свойства, связанные с выделением частей сводной таблицы.

Свойство DataRange

Свойство DataRange возвращает объект Range, представляющий диапазон области данных, занятый элементом сводной таблицы, который может располагаться в области строк, столбцов или страниц. Для иллюстрации рассмотрим код:

```
ActiveSheet.PivotTables("Sales&Trans"). _
    PivotFields("Store Type"). _
    DataRange.Select
```

Результат его выполнения показан на рис. 20.26.

Свойство LabelRange

Свойство LabelRange возвращает объект Range, представляющий диапазон области данных, занятый меткой элемента сводной таблицы, который может располагаться в области строк, столбцов или страниц. Для иллюстрации рассмотрим код:

```
ActiveSheet.PivotTables("Sales&Trans"). _
    PivotFields("Store Type"). _
    LabelRange.Select
```

В результате его применения будет выделена кнопка с меткой Store Type с рис. 20.26.

Перемещение полей сводной таблицы

Объект PivotField имеет ряд свойств, запрещающих пользователю перемещать определенные поля. Вот их список (по умолчанию все они имеют значение True):

- присвоение свойству DragToColumn значения False блокирует возможность перемещения поля в область столбцов с помощью мыши;

- присвоение свойству DragToHide значения False блокирует возможность перемещения поля за пределы сводной таблицы;
- присвоение свойству DragToPage значения False блокирует возможность перемещения поля в область страниц;
- присвоение свойству DragToRow значения False блокирует возможность перемещения поля в область строк.

	A	B	C	D	E	F	G
1	Year	(All) ▼					
2							
3				Period			
4	Store City	Store Type	Data	1	2	3	4
5	BOSTON	Company	Trans	28248	28714	28672	28602
6			Sale	44678	46927	46256	46223
7		Franchise	Trans	13993	13942	13275	13210
8			Sale	21816	21739	18632	18325
9	LOS ANGELES	Company	Trans	34588	35938	35692	35001
10			Sale	123478	129564	126340	125418
11		Franchise	Trans	71583	72947	75619	74392
12			Sale	262431	268274	279325	274531
13	NEW YORK	Company	Trans	24616	26104	27015	26854
14			Sale	78089	85251	87905	88058
15		Franchise	Trans	53273	54351	51822	51218
16			Sale	176353	182461	170537	167446

Рис. 20.26. Результат применения свойства DataRange к полю Store Type

Свойства Name, Value и SourceName

Редактируемое свойство Name возвращает или задает имя объекта PivotField. Именно это значение появляется в качестве метки выбранного поля. Свойство Value имеет тот же самый смысл, что и свойство Name.

Предназначенное только для чтения свойство SourceName возвращает имя поля в исходном источнике данных. Полученное таким способом значение может отличаться от получаемого с помощью свойства Name, если последнее было изменено.

Группировка

Приложение Excel также позволяет группировать (и разгруппировывать) данные для выделения элементов сводной таблицы в виде единого нового элемента. Для этой операции используются методы Group и Ungroup объекта Range. Обратите внимание на то, что эти методы применяются к объекту Range, а не к объекту PivotField или PivotItem.

Метод Group имеет два варианта синтаксиса, но мы рассмотрим только наиболее гибкий из них. (Более подробную информацию по этому поводу можно найти в справочной системе Excel).

Приведем небольшой пример. В применении к таблице с рис. 20.10 следующий код выделяет все данные и метки для Бостона и Нью-Йорка, объединяя их затем в единую группу, которой присваивается название Eastern. Результаты вычислений показаны на рис. 20.27. Учтите, что Excel создает новую сводную таблицу и новый элемент этой таблицы. Новое поле называется Store City2 и содержит существующий элемент Los Angeles, а также новый элемент, полученный при группировке, который называется Eastern. Без явного указания нового имени приложение присвоило бы этому элементу системное имя Group1.

```
With ActiveSheet.PivotTables("Sales&Trans")
    .PivotSelect "'Store City'[BOSTON,'New York']". xlDataAndLabel
    Set rng = Selection
    rng.Group
    .PivotFields("Store City2").PivotItems(1). Name = "Eastern"
    .PivotSelect "Eastern". xlDataAndLabel
End With
```

	A	B	C	D	E	F	G	H
1	Year	(All) ▼						
2								
3					Period			
4	Store City2	Store City	Store	Data	1	2	3	4
5	Eastern	BOSTON	Comp	Trans	28248	28714	28672	28602
6				Sale	44678	46927	46256	46223
7			Franch	Trans	13993	13942	13275	13210
8				Sale	21816	21739	18632	18325
9		NEW YORK	Comp	Trans	24616	26104	27015	26854
10				Sale	78089	85251	87905	88058
11			Franch	Trans	53273	54351	51822	51218
12				Sale	176353	182461	170537	167446
13	LOS ANGELES	LOS ANGELES	Comp	Trans	34588	35938	35692	35001
14				Sale	123478	129564	126340	125418
15			Franch	Trans	71583	72947	75619	74392
16				Sale	262431	268274	279325	274531

Рис. 20.27. Иллюстрация работы метода Group

Вычисление поля данных

Обычно в поле данных отображается значение, основанное на свойстве Function этого поля. С другой стороны, Excel разрешает пользователю изменять величину значения, показываемого в поле данных.

Свойство Calculation

Для изменения значения в поле данных достаточно задать свойство Calculation этого поля. (Настоящее свойство применимо исключительно к полям данных.) Оно может принимать значения констант из следующего перечисления:

```
Enum XlPivotFieldCalculation
    xlNoAdditionalCalculation = -4143
```



```
xlDifferenceFrom = 2  
xlPercentOf = 3  
xlPercentDifferenceFrom = 4  
xlRunningTotal = 5  
xlPercentOfRow = 6  
xlPercentOfColumn = 7  
xlPercentOfTotal = 8  
xlIndex = 9
```

End Enum

Как видно из этих констант, основная идея состоит в замене исходного значения в поле данных типом относительного значения. Чуть позднее это будет пояснено на примере.

Имейте в виду, что для некоторых значений свойства Calculation требуется дополнительная информация. В частности, если свойство Calculation имеет значение xlDifferenceFrom, xlPercentDifferenceFrom или xlPercentOf, нужно определить два следующих свойства:

- свойство BaseField, применимое только к полям данных, определяет поле сводной таблицы, используемое при вычислениях, задаваемых значениями xlDifferenceFrom, xlPercentDifferenceFrom или xlPercentOf свойства Calculation;
- свойство BaseItem, применимое только к полям данных, определяет элемент поля сводной таблицы, используемый при вычислениях, задаваемых значениями xlDifferenceFrom, xlPercentDifferenceFrom или xlPercentOf свойства Calculation.

Поясним значения констант перечисления, которые требуют задания свойств BaseField/BaseItem:

- xlDifferenceFrom отображает абсолютную разность между значением в поле сводной таблицы и базисным значением в базисном поле (оно определяется свойством BaseField);
- xlPercentOf отображает значение поля сводной таблицы, выраженное в процентах от базисного элемента в базисной ячейке;
- xlPercentDifferenceFrom — отличие этого вида вычислений от вычислений для значения константы xlDifferenceFrom состоит лишь в том, что отображается не абсолютная, а выраженная в процентах разность.

Вычисления, не требующие задания свойств BaseField/BaseItem

Перечислим значения констант перечисления, которые может принимать свойство Calculation. Эти константы не требуют указания значений свойства BaseField/BaseItem:

- xlRunningTotal — при переходе вниз по строкам таблицы рассчитывается скользящая сумма всех отображаемых значений;
- xlPercentOfRow — значения в строках пересчитываются в проценты от суммы всех значений в каждой строке;
- xlPercentOfColumn — значения в столбцах пересчитываются в проценты от суммы всех значений в каждом столбце;
- xlPercentOfTotal — значения в таблице пересчитываются в проценты от суммы всех значений в таблице.

Значение ячейки рассчитывается по следующей формуле: новое значение = (текущее значение × общая сумма)/(общая сумма по строкам × общая сумма по столбцам).

На рис. 20.28–20.31 показаны результаты этих вычислений на основе сводной таблицы с рис. 20.10. Обратите внимание на то, что вычисления процентов требуют отображения суммы данных в строках и столбцах. Вот код, создающий таблицу с рис. 20.28:

```
ActiveSheet.PivotTables("Sales&Trans").  
PivotFields("Sale").Calculation = xlRunningTotal
```

	A	B	C	D	E	F	G
1	Year	(All) ▼					
2							
3				Period			
4	Store	Store Type	Data	1	2	3	4
5	BOST	Company	Trans	28248	28714	28672	28602
6			Sale	44678	46927	46256	46223
7		Franchise	Trans	13993	13942	13275	13210
8			Sale	21816	21739	18632	18325
9	LOS	Company	Trans	34588	35938	35692	35001
10			Sale	168156	176491	172596	171641
11		Franchise	Trans	71583	72947	75619	74392
12			Sale	284247	290013	297957	292856
13	NEW	Company	Trans	24616	26104	27015	26854
14			Sale	246245	261742	260501	259699
15		Franchise	Trans	53273	54351	51822	51218
16			Sale	460600	472474	468494	460302

Рис. 20.28. Свойство Calculation имеет значение xlRunningTotal

На рис. 20.28 ячейка D6 содержит информацию о продажах в Бостоне, ячейка D10 — информацию о продажах в Бостоне и Лос-Анджелесе, а ячейка D14 — информацию о продажах в Бостоне, Лос-Анджелесе и Нью-Йорке. Впрочем, согласно некоторым отзывам, свойство Calculation не всегда работает корректно. По некоторым причинам оно может приводить к появлению в ячейках символа #N/A. Я действительно сталкивался с этим время от времени, но так и не понял причину данного явления.

Вот код, генерирующий таблицу, которая изображена на рис. 20.29:

```
ActiveSheet.PivotTables("Sales&Trans").  
PivotFields("Sale").Calculation = xlPercentOfRow
```

Код, генерирующий таблицу, которая изображена на рис. 20.30:

```
ActiveSheet.PivotTables("Sales&Trans").  
PivotFields("Sale").Calculation = xlPercentOfColumn
```

Код, генерирующий таблицу, которая изображена на рис. 20.31:

```
ActiveSheet.PivotTables("Sales&Trans").  
PivotFields("Sale").Calculation = xlPercentOfTotal
```


	A	B	C	D	E	F	G	H
1	Year	(All) ▼						
2								
3				Period				
4	Store City	Store Type	Data	1	2	3	4	Grand Total
5	BOSTON	Franchise	Trans	13993	13942	13275	13210	54420
6			Sale	27.10%	27.00%	23.14%	22.76%	100.00%
7		Company	Trans	28248	28714	28672	28602	114236
8			Sale	24.27%	25.49%	25.13%	25.11%	100.00%
9	LOS ANGELES	Company	Trans	34588	35938	35692	35001	141219
10			Sale	24.46%	25.67%	25.03%	24.85%	100.00%
11		Franchise	Trans	71583	72947	75619	74392	294541
12			Sale	24.20%	24.74%	25.75%	25.31%	100.00%
13	NEW YORK	Company	Trans	24616	26104	27015	26854	104589
14			Sale	23.01%	25.13%	25.91%	25.95%	100.00%
15		Franchise	Trans	53273	54351	51822	51218	210664
16			Sale	25.31%	26.19%	24.47%	24.03%	100.00%

Рис. 20.29. Свойство Calculation имеет значение xlPercentOfRow

	A	B	C	D	E	F	G	H
1	Year	(All)						
2								
3			Period					
4	Store	Store Type	Data	1	2	3	4	Grand Total
5	BOST	Company	Trans	28248	28714	28672	28602	114236
6			Sale	6.32%	6.39%	6.35%	6.42%	6.37%
7		Franchise	Trans	13993	13942	13275	13210	54420
8			Sale	3.09%	2.96%	2.56%	2.55%	2.79%
9	LOS	Company	Trans	34588	35938	35692	35001	141219
10			Sale	17.47%	17.65%	17.33%	17.42%	17.47%
11		Franchise	Trans	71583	72947	75619	74392	294541
12			Sale	37.13%	36.54%	38.32%	38.13%	37.53%
13	NEW	Company	Trans	24616	26104	27015	26854	104589
14			Sale	11.05%	11.61%	12.06%	12.23%	11.74%
15		Franchise	Trans	53273	54351	51822	51218	210664
16			Sale	24.95%	24.85%	23.39%	23.26%	24.11%
17	Total Trans			226301	231996	232095	229277	919669
18	Total Sale			100.00%	100.00%	100.00%	100.00%	100.00%
19								

Рис. 20.30. Свойство Calculation имеет значение xlPercentOfColumn

Вычисления, требующие задания свойств BaseField/ BaseItem

В сопроводительной документации процедура вычислений с заданием свойств BaseField/BaseItem объяснена не очень подробно, поэтому рассмотрим ее на примере:


```
With ActiveSheet.PivotTables("Sales&Trans"). _
    PivotFields("Sale")
    .Calculation = xlDifferenceFrom
    .BaseField = "Store City"
    .BaseItem = "Boston"
End With
```

	A	B	C	D	E	F	G	H
1	Year	(All) ▼						
2								
3				Period				
4	Store	Store Type	Data	1	2	3	4	Grand Total
5	BOS	Company	Trans	28248	28714	28672	28602	114236
6			Sale	1.55%	1.62%	1.60%	1.60%	6.37%
7		Franchise	Trans	13993	13942	13275	13210	54420
8			Sale	0.75%	0.75%	0.64%	0.63%	2.79%
9	LOS	Company	Trans	34588	35938	35692	35001	141219
10			Sale	4.27%	4.48%	4.37%	4.34%	17.47%
11		Franchise	Trans	71583	72947	75619	74392	294541
12			Sale	9.08%	9.28%	9.67%	9.50%	37.53%
13	NEW	Company	Trans	24616	26104	27015	26854	104589
14			Sale	2.70%	2.95%	3.04%	3.05%	11.74%
15		Franchise	Trans	53273	54351	51822	51218	210664
16			Sale	6.10%	6.31%	5.90%	5.79%	24.11%
17	Total Trans			226301	231996	232095	229277	919669
18	Total Sale			24.46%	25.40%	25.22%	24.91%	100.00%

Рис. 20.31. Свойство Calculation имеет значение xlPercentOfTotal

На рис. 20.32 воспроизведена обычная сводная таблица, изображенная на рис. 20.10, в которую был внесен ряд изменений. Во-первых, были удалены значения поля Trans, так как в данном случае они несущественны. Кроме того, значения поля Sale были заменены символами (b1–b8) для свойств BaseField и BaseItem. Ну и наконец, остальные значения поля Sale заменили знаками числа, так как их величина для данного случая несущественна.

Теперь, чтобы посмотреть, как же выполняются вычисления, нужно зафиксировать значения полей, не являющихся базовыми. В нашем случае это поля Store Type и Period. Например, присвоим им значения:

```
Store Type = "Company"
Period = 1
```

Соответствующие этим значениям ячейки с данными по продажам (поле Sale) на рис. 20.32 выделены серым цветом. Одна из этих ячеек (D4) является базовой. Для комбинации Store Type/Period вычисление осуществляется с использованием значения в этой ячейке в качестве базового. В нашем случае свойство Calculation имеет значение xlDifferenceFrom. Следовательно, базовое значение вычитается из значений в трех ячейках, выделенных серым цветом. В результате получаем таблицу, показанную на рис. 20.33. Обратите внимание: базовое значение, вычитаете-

мое само из себя, дает ноль. Это сделано для каждой комбинации Store Type/Period, как показано на рис. 20.33.

	A	B	C	D	E	F	G
1				Period			
2	Store City	Store Type	Data	1	2	3	4
3	BOSTON	Company	Trans				
4			Sale	b1	b2	b3	b4
5		Franchise	Trans				
6			Sale	b5	b6	b7	b8
7	LOS ANGE	Company	Trans				
8			Sale	#	#	#	#
9		Franchise	Trans				
10			Sale	#	#	#	#
11	NEW YOR	Company	Trans				
12			Sale	#	#	#	#
13		Franchise	Trans				
14			Sale	#	#	#	#

Рис. 20.32. Иллюстрация процесса вычислений

	A	B	C	D	E	F	G
1				Period			
2	Store City	Store Type	Data	1	2	3	4
3	BOSTON	Company	Trans				
4			Sale	0	0	0	0
5		Franchise	Trans				
6			Sale	0	0	0	0
7	LOS ANGE	Company	Trans				
8			Sale	#-b1	#-b2	#-b3	#-b4
9		Franchise	Trans				
10			Sale	#-b5	#-b6	#-b7	#-b8
11	NEW YOR	Company	Trans				
12			Sale	#-b1	#-b2	#-b3	#-b4
13		Franchise	Trans				
14			Sale	#-b5	#-b6	#-b7	#-b8

Рис. 20.33. Вид таблицы после завершения вычислений

Вот формулы для свойства Calculation, требующие задания значений свойств BaseField/BaseItem:

Значение свойства Calculation	Формула
xlDifferenceFrom	# - базовое значение
xlPercentOf	#/базовое значение (выраженное в процентах)
xlPercentDifferenceFrom	(# - базовое значение)/базовое значение (выраженное в процентах)

Для иллюстрации рассмотрим код:

```
With ActiveSheet.PivotTables("Sales&Trans"). _
    PivotFields("Sale")
    .Calculation = xlDifferenceFrom
    .BaseField = "Store City"
    .BaseItem = "Boston"
End With
```

превращающий таблицу с рис. 20.10 в таблицу, показанную на рис. 20.34.

	A	B	C	D	E	F	G
1	Year	(All)					
2							
3				Period			
4	Store	Store Type	Data	1	2	3	4
5	BOS	Company	Trans	28248	28714	28672	28602
6			Sale				
7		Franchise	Trans	13993	13942	13275	13210
8			Sale				
9	LOS	Company	Trans	34588	35938	35692	35001
10			Sale	78800	82637	80084	79195
11		Franchise	Trans	71583	72947	75619	74392
12			Sale	240615	246535	260693	256206
13	NEW	Company	Trans	24616	26104	27015	26854
14			Sale	33411	38324	41649	41835
15		Franchise	Trans	53273	54351	51822	51218
16			Sale	154537	160722	151905	149121

Рис. 20.34. Иллюстрация действия свойства Calculation

Свойство CurrentPage

Свойство CurrentPage возвращает или задает текущую страницу. Оно применимо только к полям страниц и может принимать или значение имени поля страницы, или значение All. В последнем случае показываются все страницы.

Свойство DataType

Предназначенное только для чтения свойство DataType возвращает константы, описывающие тип данных в сводной таблице. Оно может принимать значения констант из следующего перечисления:

```
Enum XlPivotFieldType
    xlText = -4158      ' Текст
    xlNumber = -4145    ' Число
    xlDate = 2          ' Дата
End Enum
```

Свойства HiddenItems и VisibleItems

Свойство HiddenItems возвращает семейство PivotItems всех скрытых объектов сводной таблицы. Вот его синтаксис:

```
PivotFieldObject.HiddenItems
```


При следующей форме записи оно может возвращать один скрытый объект `PivotItem` или их массив:

```
PivotFieldObject.HiddenItems(Index)
```

где параметр *Index* представляет собой один индекс или их массив.

Соответственно, свойство `VisibleItems` возвращает семейство `PivotItems` всех видимых объектов сводной таблицы. В качестве возвращаемого значения также может фигурировать как единичный объект, так и их массив. Синтаксис этого свойства такой же, как и в случае свойства `HiddenItems`.

Свойство `MemoryUsed`

Предназначенное только для чтения свойство `MemoryUsed` возвращает размер оперативной памяти, занятой в данный момент объектом `PivotField` (оно также применимо к объектам `PivotItem`). Возвращаемое значение принадлежит к типу `Long integer`.

Свойство `ServerBased`

Редактируемое логическое свойство `ServerBased` применимо исключительно к полям данных и используется с внешними источниками данных (обычно очень большими). Если сводная таблица построена на основе внешнего источника данных, это свойство имеет значение `True`. При этом отыскиваются только элементы, совпадающие с выделенными полями страниц.

После присвоения настоящему свойству значения `True` кэш сводной таблицы (о нем мы поговорим чуть позже) должен вмещать только данные для одного поля страниц. Это может быть важно и даже необходимо при большом размере внешнего источника данных. Недостаток состоит в том, что при каждом обновлении области страницы возможна задержка, связанная с тем, что Excel требует от внешнего источника восстановления данных на новой странице.

Существуют также ситуации, когда свойству `ServerBased` невозможно присвоить значение `True`:

- поле является групповым;
- источник данных не является внешним;
- кэш используется несколькими сводными таблицами;
- в поле содержатся данные типа, который не может находиться на стороне сервера (например, поле комментария или OLE-объект).

Свойство `ShowAllItems`

Когда редактируемое логическое свойство `ShowAllItems` получает значение `True`, в сводной таблице отображаются все элементы. По умолчанию оно имеет значение `False`, при котором скрыты элементы сводной таблицы, не связанные с полями данных. Это свойство соответствует флажку `Show items with no data` (Включать скрытые значения) в диалоговом окне `PivotTable Field` (Параметры сводной таблицы).

Предположим, что к таблице с исходными данными была добавлена еще одна строка:

1998 1 B0-1 BOSTON AStoreType 1000 \$10000.00

Эта строка соответствует новому типу магазина, но добавляет данные только для Бостона. Появившаяся в результате сводная таблица будет иметь заданный по умолчанию вид, показанный на рис. 20.35. Так как свойство ShowAllItems имеет значение False, строки, соответствующие магазину нового типа для Лос-Анджелеса или Нью-Йорка, отображаться не будут.

	A	B	C	D	E	F	G
1	Year	(All) ▼					
2							
3				Period			
4	Store	Store Type	Data	1	2	3	4
5	BOSTON	AStoreType	Trans	1000			
6			Sale	10000			
7		Company	Trans	28248	28714	28672	28602
8			Sale	44678	46927	46256	46223
9		Franchise	Trans	13993	13942	13275	13210
10			Sale	21816	21739	18632	18325
11	LOS ANGELES	Company	Trans	34588	35938	35692	35001
12			Sale	123478	129564	126340	125418
13		Franchise	Trans	71583	72947	75619	74392
14			Sale	262431	268274	279325	274531
15	NEW YORK	Company	Trans	24616	26104	27015	26854
16			Sale	78089	85251	87905	88058
17		Franchise	Trans	53273	54351	51822	51218
18			Sale	176353	182461	170537	167446
19							

Рис. 20.35. Свойство ShowAllItems имеет значение False

С другой стороны, код:

```
ActiveSheet.PivotTables("Sales&Trans").  
PivotFields("Store Type").ShowAllItems = True
```

приведет к появлению таблицы, показанной на рис. 20.36.

Метод Subtotals

Метод Subtotals позволяет рассчитать до 11 промежуточных итогов в сводной таблице. Несмотря на то что этот метод применим исключительно к полям из областей строк, столбцов и страниц, результаты отображаются только в полях строк и столбцов. Вот синтаксис этого метода:

```
PivotFieldObject.Subtotals(Index),
```

где необязательный параметр Index задает тип промежуточного итога и соответствует значению, приведенному в табл. 20.4.

	A	B	C	D	E	F	G
1	Year	(All) ▼					
2							
3				Period			
4	Store	Store Type	Data	1	2	3	4
5	BOST	AStoreType	Trans	1000			
6			Sale	10000			
7		Company	Trans	28248	28714	28672	28602
8			Sale	44678	46927	46256	46223
9		Franchise	Trans	13993	13942	13275	13210
10			Sale	21816	21739	18632	18325
11	LOS	AStoreType	Trans				
12			Sale				
13		Company	Trans	34588	35938	35692	35001
14			Sale	123478	129564	126340	125418
15		Franchise	Trans	71583	72947	75619	74392
16			Sale	262431	268274	279325	274531
17	NEW	AStoreType	Trans				
18			Sale				
19		Company	Trans	24616	26104	27015	26854
20			Sale	78089	85251	87905	88058
21		Franchise	Trans	53273	54351	51822	51218
22			Sale	176353	182461	170537	167446

Рис. 20.36. Свойство ShowAllItems имеет значение True

Таблица 20.4. Значения параметра Index метода Subtotals

Значение	Тип промежуточного итога
1	Автоматический итог (Automatic)
2	Сумма (Sum)
3	Количество значений (Count)
4	Среднее (Average)
5	Максимум (Max)
6	Минимум (Min)
7	Произведение (Product)
8	Количество численных значений (Count Nums)
9	Смещенное отклонение (StdDev)
10	Несмещенное отклонение (StdDevp)
11	Смещенная дисперсия (Var)
12	Несмещенная дисперсия (Varp)

Вот пример кода, реализующего отображение промежуточных итогов Сумма и Количество значений:

```
ActiveSheet.PivotTables("Sales&Trans").  
PivotFields("Store City").Subtotals(2) = True
```



```
ActiveSheet.PivotTables("Sales&Trans").  
PivotFields("Store City").Subtotals(3) = True
```

Можно определить свойство `Subtotals` в виде массива из 12 логических значений, что позволяет задавать несколько промежуточных итогов одновременно. Вот пример кода, приводящего к отображению всех промежуточных итогов:

```
ActiveSheet.PivotTables("Sales&Trans").  
PivotFields("Store City").  
Subtotals = Array(False, True, True, True, True,  
True, True, True, True, True, True, True)
```

Обратите внимание на то, что параметр `Automatic` в этом массиве имеет значение `False`, так как если присвоить ему значение `True`, все остальные параметры автоматически получат значение `False`.

При пропуске этого аргумента метод `Subtotals` возвращает массив логических значений для каждого промежуточного итога.

Объект PivotCache

Сводные таблицы могут влиять на источник данных различными способами, но это требует высокой вычислительной мощности. Для максимальной эффективности данные из источника помещаются в оперативную память, а сводная таблица отображает их различными способами. Это позволяет управлять данными без дальнейшего доступа к их источнику, который вполне может быть внешним.

Объект `PivotCache` представляет собой кэш сводной таблицы, то есть буферную область в оперативной памяти. Он возвращается методом `PivotCache` объекта `PivotTable`. Рассмотрим основные свойства и методы объекта `PivotCache`.

Обновление кэша сводной таблицы

Метод `Refresh` обновляет кэш и сводную таблицу.

Однако существуют способы предотвратить обновление сводной таблицы. Во-первых, это можно сделать с помощью пользовательского интерфейса. Достаточно выбрать пункт `Refresh data` (Обновить данные) в меню `PivotTable` (Сводные таблицы). Во-вторых, это делается программным способом путем присвоения свойству `EnableRefresh` значения `False`.

Если редактируемое логическое свойство `RefreshOnFileOpen` имеет значение `True`, кэш сводной таблицы автоматически обновляется при открытии книги пользователем. По умолчанию это свойство имеет значение `False`. Если свойству `EnableRefresh` присвоить значение `False`, свойство `RefreshOnFileOpen` игнорируется. Кроме того, имейте в виду, что даже если свойство `RefreshOnFileOpen` имеет значение `True`, автоматического обновления кэша сводной таблицы при открытии книги не происходит.

Свойство `RefreshDate` возвращает дату и время последнего обновления кэша, а свойство `RefreshName` — имя пользователя, выполнившего последнее обновление.

Свойство MemoryUsed

Предназначенное только для чтения свойство MemoryUsed применимо к объектам PivotCache и PivotField и возвращает число байт оперативной памяти, используемой кэшем в данный момент.

Свойство OptimizeCache

Редактируемое логическое свойство OptimizeCache используется для оптимизации работы кэша с очень большими и сложными массивами данных. Оптимизация замедляет выполнение начала запроса и построение буфера. Ей соответствует значение True, ее блокировке — False. По умолчанию свойство OptimizeCache имеет значение False. К сожалению, в документации нет информации о точных последствиях оптимизации кэша, поэтому решить, стоит ли ее применять, можно только путем проб и ошибок. Соответственно, при отсутствии явных проблем с производительностью лучше всего оставить свойству OptimizeCache заданное по умолчанию значение False.

Свойство RecordCount

Доступное только для чтения свойство RecordCount возвращает текущее число записей, помещенных в кэш.

Свойство SourceData

Свойство SourceData возвращает источник данных сводной таблицы, как уже упоминалось в разделе, посвященном объекту PivotTable.

Свойство Sql

Редактируемое строковое свойство Sql возвращает или задает строку запроса SQL, используемую при работе с источниками данных ODBC.

Объект PivotItem

Объект PivotItem применяется для представления уникальных элементов в поле сводной таблицы. Для иллюстрации рассмотрим следующий код:

```
Dim pi As PivotItem
For Each pi In
    ActiveSheet.PivotTables("Sales&Trans").
        PivotFields("Store City").PivotItems
    Debug.Print pi.Name
Next
```

В результате его выполнения появится список:

```
BOSTON
LOS ANGELES
NEW YORK.
```

который содержит различные значения поля Store City.

Метод `PivotItems` объекта `PivotField` возвращает объекты `PivotItem`. Запись: `PivotFieldObject.PivotItems` возвращает набор всех объектов `PivotItem` для указанного поля сводной таблицы. Если же написать:

`PivotFieldObject.PivotItems(Index)`,
возвращен будет один объект `PivotItem` или же массив объектов `PivotItem` (при установке `Index` в массив индексов).

В табл. 20.5 перечислены свойства и методы объекта `PivotItem`. О некоторых из них мы поговорим более подробно. Имейте в виду, что к объекту `PivotItem` применимы также некоторые свойства и методы объекта `PivotField`.

Таблица 20.5. Свойства и методы объекта `PivotItem`

Application	IsCalculated	Position
ChildItems	LabelRange	RecordCount
Creator	Name	ShowDetail
DataRange	Parent	SourceName
Delete	ParentItem	Value
Formula	ParentShowDetail	Visible

Свойство `DataRange`

Свойство `DataRange` возвращает объект `Range`, представляющий область данных, занятую рассматриваемым объектом `PivotItem`. Для иллюстрации рассмотрим код:

```
ActiveSheet.PivotTables("Sales&Trans"). _  
    PivotFields("Store Type"). _  
    PivotItems("Company").DataRange.Select,  
результат выполнения которого показан на рис. 20.37.
```

	A	B	C	D	E	F	G
1	Year	(All) ▼					
2							
3				Period			
4	Store City	Store Type	Data	1	2	3	4
5	BOSTON	Company	Trans	28248	28714	28672	28602
6			Sale	44678	46927	46256	46223
7		Franchise	Trans	13993	13942	13275	13210
8			Sale	21816	21739	18632	18325
9	LOS ANGELES	Company	Trans	34588	35938	35692	35001
10			Sale	123478	129564	126340	125418
11		Franchise	Trans	71583	72947	75619	74392
12			Sale	262431	268274	279325	274531
13	NEW YORK	Company	Trans	24616	26104	27015	26854
14			Sale	78089	85251	87905	88058
15		Franchise	Trans	53273	54351	51822	51218
16			Sale	176353	182461	170537	167446

Рис. 20.37. Свойство `DataRange` для поля `Store Type` имеет значение `Company`

Свойство LabelRange

Свойство LabelRange возвращает объект Range, соответствующий меткам рассматриваемого элемента в сводной таблице. На рис. 20.38 показан результат выполнения кода:

```
ActiveSheet.PivotTables("Sales&Trans"). _
    PivotFields("Store Type"). _
    PivotItems("Company").LabelRange.Select
```

	A	B	C	D	E	F	G
1	Year	(All) ▼					
2							
3				Period			
4	Store City	Store Type	Data	1	2	3	4
5	BOSTON	Company	Trans	28248	28714	28672	28602
6			Sale	44678	46927	46256	46223
7		Franchise	Trans	13993	13942	13275	13210
8			Sale	21816	21739	18632	18325
9	LOS ANGELES	Company	Trans	34588	35938	35692	35001
10			Sale	123478	129564	126340	125418
11		Franchise	Trans	71583	72947	75619	74392
12			Sale	262431	268274	279325	274531
13	NEW YORK	Company	Trans	24616	26104	27015	26854
14			Sale	78089	85251	87905	88058
15		Franchise	Trans	53273	54351	51822	51218
16			Sale	176353	182461	170537	167446

Рис. 20.38. Свойство LabelRange для поля Store Type имеет значение Company

Свойство IsCalculated

Свойство IsCalculated возвращает значение True, если элемент сводной таблицы является вычисляемым. О вычисляемых элементах мы поговорим чуть позднее.

Свойства Name, Value и SourceName

Редактируемое свойство Name возвращает или задает имя элемента сводной таблицы. Именно это значение появляется в качестве метки элемента. Свойство Value аналогично свойству Name.

Предназначенное только для чтения свойство SourceName возвращает имя элемента в исходной таблице данных. Оно может отличаться от значения свойства Name, если последнее было изменено.

Свойство Position

Свойство Position возвращает или задает положение элемента сводной таблицы. К примеру, код:


```
ActiveSheet.PivotTables("Sales&Trans").
    PivotFields("Store Type").
        PivotItems("Franchise").Position
```

возвращает число 2, так как поле Franchise является вторым элементом сводной таблицы (см. рис. 20.10). Более того, можно поменять положение полей Company и Franchise, присвоив свойству Position элемента Franchise значение 1:

```
ActiveSheet.PivotTables("Sales&Trans").
    PivotFields("Store Type").
        PivotItems("Franchise").Position = 1
```

Свойство RecordCount

Редактируемое свойство RecordCount возвращает количество записей в кэше сводной таблицы, которые содержат указанный элемент. Например, код:

```
ActiveSheet.PivotTables("Sales&Trans").
    PivotFields("Store Type").
        PivotItems("Franchise").RecordCount
```

вернет число 80, так как в исходной таблице (а следовательно, и в ее кэше) существует 80 строк, содержащих упоминание о типе магазина Franchise.

Свойство ShowDetail

Когда редактируемое свойство ShowDetail имеет значение True, элемент сводной таблицы показывается подробно; в противном случае объект PivotItem скрыт. Для иллюстрации рассмотрим, как обычно, таблицу с рис. 20.10. Код:

```
ActiveSheet.PivotTables("Sales&Trans").
    PivotFields("Store City").
        PivotItems("Boston").ShowDetail = False
```

формирует сводную таблицу, показанную на рис. 20.39. Как легко заметить, значения Transactions и Sales для Бостона суммированы по обоим типам магазинов (Company и Franchise).

	A	B	C	D	E	F	G
1	Year	(All)					
2							
3			Period				
4	Store City	Store Type	Data	1	2	3	4
5	BOSTON		Trans	42241	42656	41947	41812
6			Sale	66494	68666	64888	64548
7	LOS ANGELES	Company	Trans	34588	35938	35692	35001
8			Sale	123478	129564	126340	125418
9		Franchise	Trans	71583	72947	75619	74392
10			Sale	262431	268274	279325	274531
11	NEW YORK	Company	Trans	24616	26104	27015	26854
12			Sale	78089	85251	87905	88058
13		Franchise	Trans	53273	54351	51822	51218
14			Sale	176353	182461	170537	167446

Рис. 20.39. Иллюстрация свойства ShowDetail

К сожалению, применение метода ShowDetail к внутренним элементам сводной таблицы иногда сопровождается проблемами. Например, код:

```
ActiveSheet.PivotTables("Sales&Trans"). _
    PivotFields("Store Type"). _
    PivotItems("Company").ShowDetail = False
```

на самом деле не присваивает свойству ShowDetail значение False, что можно проверить следующим образом:

```
MsgBox ActiveSheet.PivotTables("Sales&Trans"). _
    PivotFields("Store Type"). _
    PivotItems("Company").ShowDetail
```

Сводная таблица не отразит выполненных изменений! По крайней мере, именно это происходило в двух операционных системах, в которых автор проверял данный код. Подобная проблема возникает также со свойством Subtotals.

Следующий пример кода включает режим подробного отображения элемента Boston:

```
With ActiveSheet.PivotTables("Sales&Trans"). _
    PivotFields("Store City").PivotItems("Boston") _
    .ShowDetail = Not .ShowDetail
End With
```

Свойство ShowDetail также применимо к объекту Range, даже когда диапазон принадлежит сводной таблице. Следующий пример кода также приводит к появлению таблицы с рис. 20.39:

```
ActiveSheet.PivotTables("Sales&Trans"). _
    PivotSelect "Boston", xlDataAndLabel _
    Selection.ShowDetail = False
```

А вот пример кода, включающего режим подробного отображения элемента Boston:

```
Dim rng As Range
ActiveSheet.PivotTables("Sales&Trans"). _
    PivotSelect "Boston", xlDataAndLabel _
    Set rng = Selection.Cells(1, 1)
    rng.ShowDetail = Not rng.ShowDetail
```

Переменная rng относится только к первой ячейке диапазона, представляющего данные и метку для Бостона; соответственно, мы получим:

```
Set rng = Selection.Cells(1, 1)
```

Это происходит из-за того, что в применении к объекту Range свойство ShowDetail возвращает значение только в том случае, если диапазон представлен всего одной ячейкой.

В качестве заключительного примера рассмотрим код, скрывающий подробности всего поля Store City:

```
ActiveSheet.PivotTables("Sales&Trans"). _
    PivotSelect "'Store City'", xlDataAndLabel _
    Selection.ShowDetail = False
ActiveSheet.PivotTables("Sales&Trans"). _
    PivotSelect "'Store City'", xlDataAndLabel
```

(Замена False на True приводит к отображению подробностей.)

В заключение остается заметить, что из-за недостатка информации о свойстве ShowDetails в справочной документации Excel перед его использованием желательно провести ряд экспериментов.

Свойство Visible

Свойство Visible определяет отображение элементов сводной таблицы.

Объекты PivotCell и PivotItemList

В приложении Excel XP появился новый объект PivotCell, представляющий ячейку сводной таблицы. Этот объект и его потомки показаны на рис. 20.40.

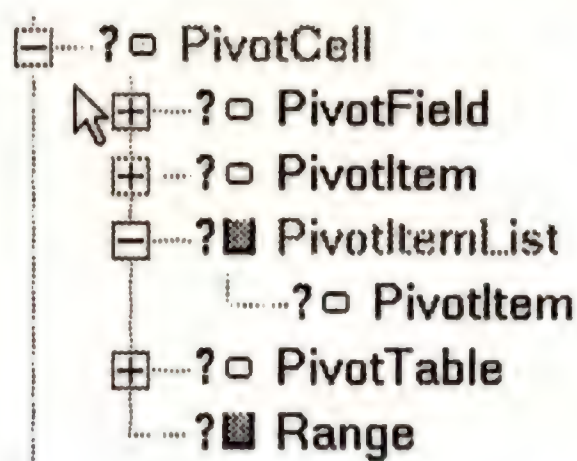


Рис. 20.40. Объект PivotCell

Для получения объекта PivotCell используется свойство PivotCell объекта Range, как в примере:

```
MsgBox ActiveSheet.Range("C5").PivotCell.PivotCellType
```

Если объект Range состоит из нескольких ячеек, возвращается только первая из них.

Объект PivotCell не имеет ни одного метода. Среди его 13 свойств можно выделить самые важные:

- ColumnItems
- CustomSubtotalFunction
- DataField
- PivotCellType
- PivotField
- PivotItem
- PivotTable
- RowItems

В следующих разделах главы будет дано краткое описание этих свойств. Важно помнить, что приложение Excel не всегда реагирует благоприятно в процессе их применения к объекту PivotCell, и может появиться сообщение об ошибке. Например, если ячейка не содержит данных, доступ к свойству ColumnItems объекта PivotCell приведет к сообщению об ошибке, вместо того чтобы вернуть пустой набор.

Свойства ColumnItems, RowItems и DataField

Свойства ColumnItems и RowItems возвращают семейство PivotItemList, содержащее столбцы (или строки) элементов сводной таблицы, связанных с объектом PivotCell.

Свойство `DataField` возвращает один элемент `PivotField`, связанный с объектом `PivotCell`. Имейте в виду, что попытка применить эти свойства к объекту `PivotCell`, который не является группой ячеек с данными в сводной таблице, приведет к появлению сообщения об ошибке.

Для иллюстрации рассмотрим сводную таблицу, изображенную на рис. 20.6. Процедура `PivotItemList` показывает элементы столбцов, строк и полей данных сводной таблицы, которые связаны с объектом `PivotCell` диапазона E5.

```
Sub PivotItemList()

' Иллюстрация свойств ColumnItems и RowItems

Dim rng As Range
Dim pi As PivotItem
Dim ws As Worksheet

Set ws = ActiveSheet

' Следующий код не будет работать, если
' переменная rng не является ячейкой с данными.
Set rng = ws.Range("E5")

Debug.Print "Элементы столбцов:"
For Each pi In rng.PivotCell.ColumnItems
    Debug.Print pi.Name
Next

Debug.Print "Элементы строк:"
For Each pi In rng.PivotCell.RowItems
    Debug.Print pi.Name
Next

Debug.Print "Поля данных:"
Debug.Print rng.PivotCell.DataField.Name

End Sub
```

В результате выполнения этой процедуры будет получен следующий результат:

```
Элементы столбцов:
2
Элементы строк:
BOSTON
Company
Поля данных:
Sum of Transactions
```

Как уже упоминалось, если изменить диапазон на ячейку C5, строка:

```
For Each pi In rng.PivotCell.ColumnItems
```

вызовет сообщение об ошибке `Application defined or object-defined error`.

Свойство `CustomSubtotalFunction`

Без сомнения, вы уже знаете, что в сводной таблице можно сделать видимыми промежуточные значения. Например, на рис. 20.6 показаны суммы по городам. Теперь тип промежуточного значения может определяться автоматически или пользователем: сумма, количество значений, среднее и т. п. Во втором случае объект `PivotCell`,

отображающий тип промежуточной суммы (но не ее значение), принадлежит к типу xlPivotCellCustomSubtotal (см. обсуждение свойства PivotCellType). Например, если изменить промежуточную сумму по городам на среднее, объект PivotCell для ячеек A9, B9 и C9 на рис. 20.6 будет принадлежать к типу xlPivotCellCustomSubtotal.

Свойство CustomSubtotalFunction объекта PivotCell возвращает тип обобщенной функции для пользовательского промежуточного значения в ячейке. Возвращаемые значения определяются следующим перечислением:

```
Enum XlConsolidationFunction
    xlVarP = -4165
    xlVar = -4164
    xlSum = -4157
    xlStDevP = -4156
    xlStDev = -4155
    xlProduct = -4149
    xlMin = -4139
    xlMax = -4136
    xlCountNums = -4113
    xlCount = -4112
    xlAverage = -4106
    xlUnknown = 1000
End Enum
```

Если объект PivotCell не принадлежит к типу xlPivotCellCustomSubtotal, любые ссылки на свойство CustomSubtotalFunction приводят к сообщению об ошибке. Поэтому важно сначала проверять свойство PivotcellType, о котором мы поговорим чуть ниже!

Свойство PivotCellType

Свойство PivotCellType возвращает тип ячейки в форме констант из следующего перечисления:

```
Enum XlPivotCellType
    xlPivotCellValue = 0           ' Значение
    xlPivotCellPivotItem = 1       ' Элемент сводной таблицы
    xlPivotCellSubtotal = 2        ' Промежуточная сумма
    xlPivotCellGrandTotal = 3      ' Общая сумма
    xlPivotCellDataField = 4       ' Поле данных
    xlPivotCellPivotField = 5      ' Поле сводной таблицы
    xlPivotCellPageFieldItem = 6   ' Элемент страницы
    xlPivotCellCustomSubtotal = 7  ' Пользовательская сумма
    xlPivotCellDataPivotField = 8  ' Поле данных сводной таблицы
    xlPivotCellBlankCell = 9       ' Пустая ячейка
End Enum
```

На рис. 20.41 показаны сводная таблица и справа от нее значения свойства PivotCellType. Это позволяет увидеть, какие типы ячеек содержатся в этой таблице. Например, ячейка A4 имеет значение свойства PivotCellType равное 5, отмечающее ячейку сводной таблицы.

Свойства PivotTable, PivotField и PivotItem

Свойство PivotTable возвращает объект PivotTable, связанный с ячейками сводной таблицы. Соответственно, свойство PivotField возвращает объект PivotField, связанный с ячейками сводной таблицы; а свойство PivotItem возвращает объект PivotItem.

Взяв за основу сводную таблицу, изображенную на рис. 20.6, проиллюстрируем эти свойства на следующем примере:

```
Sub test()  
  
    ' См. рис 20.6  
  
    Dim rng As Range  
    Dim ws As Worksheet  
  
    Set ws = ActiveSheet  
  
    Set rng = ws.Range("E5")  
    Debug.Print rng.PivotCell.PivotTable.Name    ' возвращает Sales&Trans  
    Debug.Print rng.PivotCell.PivotField        ' возвращает Sum of Transactions  
    Set rng = ws.Range("B5")  
    Debug.Print rng.PivotCell.PivotItem.Name    ' возвращает Company  
  
End Sub
```

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	Year	(All)								5	6	0	0	0	0	0
2										0	0	0	0	0	0	0
3				Period						9	9	9	5	9	9	9
4	Store C	Store Type	Data	1	2	3	4	Grand Total	5	5	8	1	1	1	1	3
5	BOSTON	Company	Sum of Trans	24.73%	25.14%	25.10%	25.04%	100.00%	1	1	4	0	0	0	0	0
6			Sum of Sales	44678	46927	46256	46223	184084	1	1	4	0	0	0	0	0
7		Franchise	Sum of Trans	25.71%	25.62%	24.39%	24.27%	100.00%	1	1	4	0	0	0	0	0
8			Sum of Sales	21816	21739	18632	18325	80512	1	1	4	0	0	0	0	0
9	BOSTON Sum of Transactions			25.05%	25.29%	24.87%	24.79%	100.00%	2	2	2	0	0	0	0	0
10	BOSTON Sum of Sales			66494	68666	64888	64548	264596	2	2	2	0	0	0	0	0
11	LOS ANG	Company	Sum of Trans	24.49%	25.45%	25.27%	24.78%	100.00%	1	1	4	0	0	0	0	0
12			Sum of Sales	123478	129564	126340	125418	504800	1	1	4	0	0	0	0	0
13		Franchise	Sum of Trans	24.30%	24.77%	25.67%	25.26%	100.00%	1	1	4	0	0	0	0	0
14			Sum of Sales	262431	268274	279325	274531	1084561	1	1	4	0	0	0	0	0
15	LOS ANGELES Sum of Transactions			24.36%	24.99%	25.54%	25.10%	100.00%	2	2	2	0	0	0	0	0
16	LOS ANGELES Sum of Sales			385909	397838	405665	399949	1589361	2	2	2	0	0	0	0	0
17	NEW YOR	Company	Sum of Trans	23.54%	24.96%	25.83%	25.68%	100.00%	1	1	4	0	0	0	0	0
18			Sum of Sales	78089	85251	87905	88058	339303	1	1	4	0	0	0	0	0
19		Franchise	Sum of Trans	25.29%	25.80%	24.60%	24.31%	100.00%	1	1	4	0	0	0	0	0
20			Sum of Sales	176353	182461	170537	167446	696797	1	1	4	0	0	0	0	0
21	NEW YORK Sum of Transactions			24.71%	25.52%	25.01%	24.76%	100.00%	2	2	2	0	0	0	0	0
22	NEW YORK Sum of Sales			254442	267712	258442	255504	1036100	2	2	2	0	0	0	0	0
23	Total Sum of Transactions			24.61%	25.23%	25.24%	24.93%	100.00%	3	3	3	0	0	0	0	0
24	Total Sum of Sales			706845	734216	728995	720001	2890057	3	3	3	0	0	0	0	0

Рис. 20.41. Значения свойства PivotCellType

Вычисляемые элементы и поля

Вы уже убедились, что в сводную таблицу можно добавить вычисляемое поле. Это специальный тип объекта PivotField, не являющийся частью исходного источника данных, а вычисляемый на основе этих данных по формуле. Обратите внимание на то, что такого объекта, как CalculatedField, не существует, зато есть семейство CalculatedFields.

Как уже упоминалось, для создания нового вычисляемого поля используется метод Add семейства CalculatedFields объекта PivotTable:

```
CalculatedFieldsObject.Add(Name, Formula).
```

где параметр Name представляет собой имя поля, а параметр Formula — формулу, по которой производятся вычисления.

С другой стороны, вычисляемый элемент является специальным типом объекта PivotItem, связанным с рассматриваемой сводной таблицей. Опять же, не существует объекта CalculatedItem, в то время как имеется семейство CalculatedItems. Значения элементов вычисляются с помощью формул.

Объект PivotField имеет семейство CalculatedItems, в которое входят все вычисляемые элементы для данного поля сводной таблицы. Для создания нового вычисляемого элемента используется метод Add объекта CalculatedItems. Синтаксис этого метода тот же, что и в предыдущем случае:

```
CalculatedItemsObject.Add(Name, Formula).
```

где параметр Name представляет собой имя поля, а параметр Formula — формулу, по которой производятся вычисления.

Для иллюстрации рассмотрим код, добавляющий новый вычисляемый элемент в поле Store Type:

```
ActiveSheet.PivotTables("Sales&Trans").  
PivotFields("Store Type").CalculatedItems.  
Add "CompanyX2", "='Store Type'Company*2"
```

	A	B	C	D	E	F	G
1	Year	(All) ▼					
2							
3				Period			
4	Store Cty	Store Type	Data	1	2	3	4
5	BOSTON	Company	Trans	28248	28714	28672	28602
6			Sale	44678	46927	46256	46223
7		Franchise	Trans	13993	13942	13275	13210
8			Sale	21816	21739	18632	18325
9		CompanyX2	Trans	56496	57428	57344	57204
10			Sale	89356	93854	92512	92446
11	LOS ANGELES	Company	Trans	34588	35938	35692	35001
12			Sale	123478	129564	126340	125418
13		Franchise	Trans	71583	72947	75619	74392
14			Sale	262431	268274	279325	274531
15		CompanyX2	Trans	69176	71876	71384	70002
16			Sale	246956	259128	252680	250836
17	NEW YORK	Company	Trans	24616	26104	27015	26854
18			Sale	78089	85251	87905	88058
19		Franchise	Trans	53273	54351	51822	51218
20			Sale	176353	182461	170537	167446
21		CompanyX2	Trans	49232	52208	54030	53708
22			Sale	156178	170502	175810	176116

Рис. 20.42. Вычисляемый элемент CompanyX2

Результат показан на рис. 20.42. Вычисляемый элемент называется CompanyX2. Значение каждой из ячеек CompanyX2 в два раза больше соответствующего значения ячейки Company.

Для сравнения добавим вычисляемое поле в сводную таблицу, изображенную на рис. 20.42. Это будет то же самое поле, что и при обсуждении метода CalculatedFields, имевшего место ранее в этой главе:

```
With ActiveSheet.PivotTables("Sales&Trans"). _  
    CalculatedFields.Add("Average", _  
        "= Sales/Transactions")  
        .Orientation = xlDataField  
        .Name = "Avg Check"  
        .NumberFormat = "##.#"  
End With
```

Результат показан на рис. 20.43.

	A	B	C	D	E	F	G
1	Year	(All) ▼					
2							
3				Period			
4	Store City	Store Type	Data	1	2	3	4
5	BOSTON	Company	Trans	28248	28714	28672	28602
6			Sale	44678	46927	46256	46223
7			Avg Check	1.6	1.6	1.6	1.6
8		Franchise	Trans	13993	13942	13275	13210
9			Sale	21816	21739	18632	18325
10			Avg Check	1.6	1.6	1.4	1.4
11		CompanyX2	Trans	56496	57428	57344	57204
12			Sale	89356	93854	92512	92446
13			Avg Check	1.6	1.6	1.6	1.6
14	LOS ANGELES	Company	Trans	34588	35938	35692	35001
15			Sale	123478	129564	126340	125418
16			Avg Check	3.6	3.6	3.5	3.6
17		Franchise	Trans	71583	72947	75619	74392
18			Sale	262431	268274	279325	274531
19			Avg Check	3.7	3.7	3.7	3.7
20		CompanyX2	Trans	69176	71876	71384	70002
21			Sale	246956	259128	252680	250836
22			Avg Check	3.6	3.6	3.5	3.6
23	NEW YORK	Company	Trans	24616	26104	27015	26854
24			Sale	78089	85251	87905	88058
25			Avg Check	3.2	3.3	3.3	3.3
26		Franchise	Trans	53273	54351	51822	51218
27			Sale	176353	182461	170537	167446
28			Avg Check	3.3	3.4	3.3	3.3
29		CompanyX2	Trans	49232	52208	54030	53708
30			Sale	156178	170502	175810	176116
31			Avg Check	3.2	3.3	3.3	3.3

Рис. 20.43. Вычисляемое поле и вычисляемый элемент

Метод ListFormulas объекта PivotTable приведет к появлению на отдельном листе списка всех формул, используемых в сводной таблице. Результат для сводной таблицы, изображенной на рис. 20.43, показан на рис. 20.44.

Напоследок напомним, что для определения, является ли поле сводной таблицы или ее элемент вычисляемым, используется исключительно свойство IsCalculated.

	A	B	C	D	E
1	Calculated Field				
2	Solve Order	Field	Formula		
3	1	Average	=Sales /Transactions		
4					
5	Calculated Item				
6	Solve Order	Item	Formula		
7	1	CompanyX2	=Company*2		
8					
9					
10	Note:				
11	When a cell is updated by more than one formula,				
12	the value is set by the formula with the last solve order.				
13					
14	To change formula solve orders,				
15	use the Solve Order command on the PivotTable command bar.				

Рис. 20.44. Результат применения метода ListFormulas к сводной таблице, изображенной на рис. 20.43

Пример: вывод на печать сводной таблицы

Теперь пришло время добавить в надстройку SRXUtils функцию PrintPivotTables. Сложные книги могут содержать большое количество сводных таблиц, расположенных на разных листах. Поэтому простая служебная программа для вывода на печать этих сводных таблиц будет весьма полезной. К автору неоднократно обращались с просьбами написать подобную программу.

Наше приложение приводит к появлению диалогового окна, показанного на рис. 20.45. Поле со списком содержит перечень всех сводных таблиц. Каждый пункт включает имя сводной таблицы, за которым следует имя листа. Пользователю достаточно выделить одно или несколько названий сводных таблиц и щелкнуть на кнопке Print.

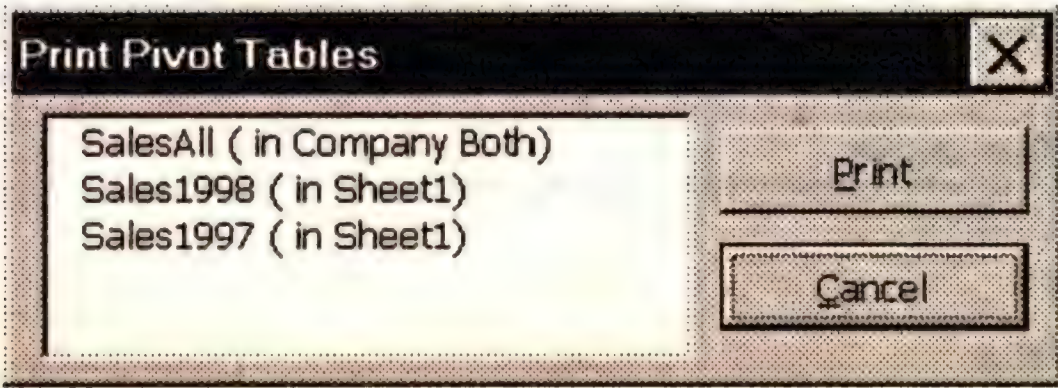


Рис. 20.45. Диалоговое окно Print Pivot Tables

Так как процесс создания служебной программы осуществляется в книге Print.xls, для начала требуется открыть эту книгу. После внесения изменений потребуется также сохранить эту книгу как Print.utl. Если надстройка Print.utl уже загружена, единственным способом выгрузить ее является снятие флажка SRXUtils.xla и закрытие книги SRXUtils.xls.

Создание объекта UserForm

Создайте в книге Print.xls диалоговое окно, показанное на рис. 20.45. Присвойте свойству Name этого окна имя dlgPrintPivotTables, а свойству Caption — имя Print Pivot Tables. Код процедуры PrintPivotTables показан в листинге 20.5.

Листинг 20.5. Процедура PrintPivotTables

```
Public Sub PrintPivotTables()  
    dlgPrintPivotTables.Show  
End Sub
```

Диалоговое окно dlgPrintPivotTables имеет две командные кнопки и поле со списком.

Поле со списком

Поместите поле со списком в форму, как показано на рис. 20.45. В диалоговом окне Properties установите следующие свойства:

Свойство	Значение
Name	IstPTs
TabIndex	0
MultiSelect	frmMultiSelectExtended

Свойство MultiSelect имеет значение frmMultiSelectExtended, то есть пользователь может с помощью клавиши Ctrl выделить несколько непоследовательных имен листов, а с помощью клавиши Shift — набор последовательных имен листов.

Свойство TabIndex указывает не только на последовательность активизации элементов формы при нажатии клавиши Tab, но и на элемент, который будет активен при первой загрузке формы. Так как изначально активным должно быть поле со списком, свойству TabIndex этого элемента присваивается значение 0.

Кнопка Print

Поместите командную кнопку в форму, изображенную на рис. 20.45, и в окне Properties задайте следующие свойства:

Свойство	Значение
Name	CmdPrint
Accelerator	P
Caption	Print
TabIndex	1

Кнопка Cancel

Поместите еще одну командную кнопку в форму, изображенную на рис. 20.45, и в окне Properties задайте следующие свойства:

Свойство	Значение
Name	cmdCancel
Accelerator	C
Caption	Cancel
TabIndex	2
Cancel	True

Когда свойству Cancel кнопки cmdCancel присвоено значение True, оно активизируется, когда пользователь нажимает клавишу Esc. То есть нажатие этой клавиши будет приводить к исчезновению диалогового окна.

Создание кода для объекта UserForm

Пришло время сопоставить с элементами управления некий код.

Область описаний

Область описаний должна содержать описание переменных уровня модуля, как показано в листинге 20.6.

Листинг 20.6. Описание переменных уровня модуля

```
Dim cPTs As Integer
Dim sPTNames() As String
Dim sSheets() As String
```

Код кнопки Cancel

Код, соответствующий кнопке Cancel, показан в листинге 20.7.

Листинг 20.7. Обработка события cmdCancel_Click

```
Private Sub cmdCancel_Click()
    Unload Me
End Sub
```

Код кнопки Print

Нажатие кнопки Print вызывает основную процедуру и затем выгружает форму; код соответствующего события показан в листинге 20.8.

Листинг 20.8. Обработка события cmdPrint_Click

```
Private Sub cmdPrint_Click()
    PrintSelectedPTs
    Unload Me
End Sub
```

Событие Initialize

Событие Initialize объекта UserForm должно приводить к заполнению поля со списком сводных таблиц. В нашем приложении используются два массива уровня мо-

дуля: один сохраняет имена листов, а другой — имена сводных таблиц. Существует также переменная уровня модуля, сохраняющая число сводных таблиц. Заполнение этих массивов будет связано с событием `Initialize`, как показано в листинге 20.9. Затем на основе массивов сформируется список таблиц. Массивы снова будут использоваться в основной процедуре распечатки. Именно поэтому при описании было указано, что они находятся на уровне модуля.

Обратите внимание на применение оператора `ReDim` для изменения количества элементов в массивах. Это необходимо, так как количество сводных таблиц в книге заранее неизвестно.

Листинг 20.9. Событие `Initialize`

```
Private Sub UserForm_Initialize()
' Заполнение поля со списком сводных таблиц
Dim ws As Worksheet
Dim PT As PivotTable
ReDim sPTNames(1 To 10) As String
ReDim sSheets(1 To 10) As String
lstPTs.Clear
cPTs = 0
For Each ws In ActiveWorkbook.Worksheets
    For Each PT In ws.PivotTables
        ' Update PT count
        cPTs = cPTs + 1
        ' При необходимости количество элементов массива изменяется
        If UBound(sSheets) < cPTs Then
            ReDim Preserve sSheets(1 To cPTs + 5)
            ReDim Preserve sPTNames(1 To cPTs + 5)
        End If
        ' Сохранение имени сводной таблицы и переменной ws
        sPTNames(cPTs) = PT.Name
        sSheets(cPTs) = ws.Name
        ' Добавление элемента в список
        lstPTs.AddItem PT.Name & " ( in " & _
            sSheets(cPTs) & ")"
    Next
Next
End Sub
```

Процедура `PrintPTs`

Код основной процедуры вывода на печать показан в листинге 20.10. Особое внимание требуется обратить на два момента. Во-первых, в книге может вообще не содержаться сводных таблиц. Во-вторых, пользователь может нажать кнопку `Print`, не выделив ни одного элемента в поле со списком.

Важно понимать, что все поля со списком представляют собой массивы, нумерация которых начинается с нуля, а нумерация нашего массива со списком листов книги начинается с единицы. То есть следует помнить, что члену массива номер i соответствует индекс $i + 1$.

Листинг 20.10. Процедура `PrintSelectedPTs`

```
Sub PrintSelectedPTs()
' Печать сводных таблиц, имена которых выделены в поле со списком
Dim i As Integer
Dim bNoneSelected As Boolean
bNoneSelected = True
```



```
If cPTs = 0 Then
    MsgBox "В этой книге нет сводных таблиц.", _
        vbExclamation
    Exit Sub
Else
    For i = 0 To lstPTs.ListCount - 1
        If lstPTs.Selected(i) Then
            bNoneSelected = False
            ' Нумерация в поле со списком начинается с 0.. а нумерация массива с 1
            Worksheets(sSheets(i + 1)).
                PivotTables(sPTNames(i + 1)). _
                    TableRange2.PrintOut
        End If
    Next
End If
If bNoneSelected Then
    MsgBox "Не было выделено ни одного имени сводной таблицы.", _
        vbExclamation
End If
End Sub
```


21 Объект Chart

Диаграммы в Excel представлены объектами Chart объектной модели Excel. Так как диаграмма — это достаточно непростая структура, Chart является одним из самых сложных представителей объектной модели.

На рис. 21.1 показана типичная диаграмма и несколько связанных с ней терминов.

На рис. 21.2 приводится объект Chart со своими непосредственными потомками.

Объекты Chart и ChartObject

Как вы уже, вероятно, знаете, диаграммы Excel могут располагаться как на отдельном листе, называемом *лист диаграммы*, так и встраиваться в обычный лист с данными. Соответственно, объект Chart может представлять собой лист диаграммы или встроенную диаграмму. В последнем случае он не включается непосредственно в состав листа. Скорее, это лист, содержащий объект ChartObject, который выполняет роль контейнера для диаграммы.

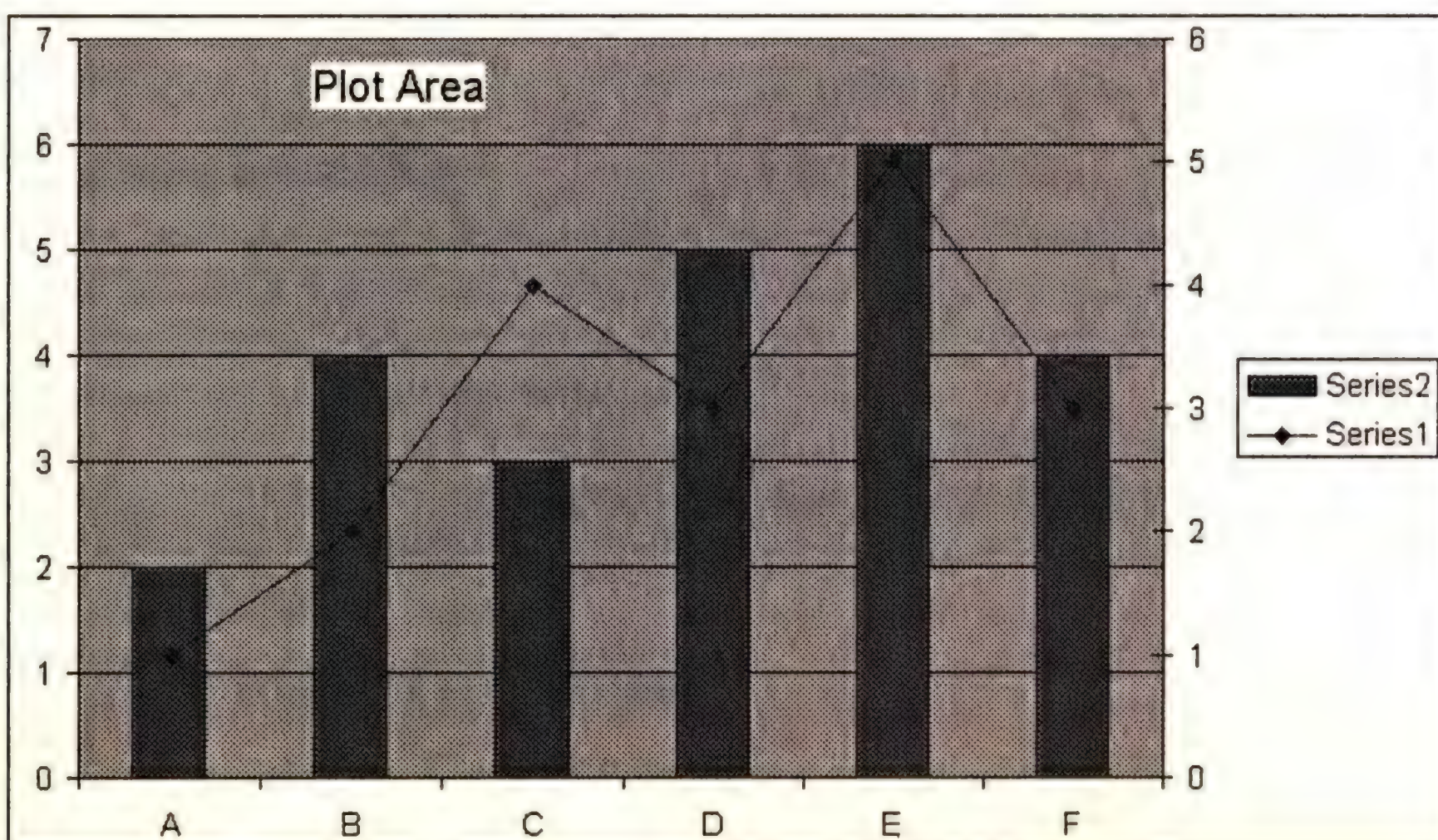


Рис. 21.1. Диаграмма и связанные с ней термины

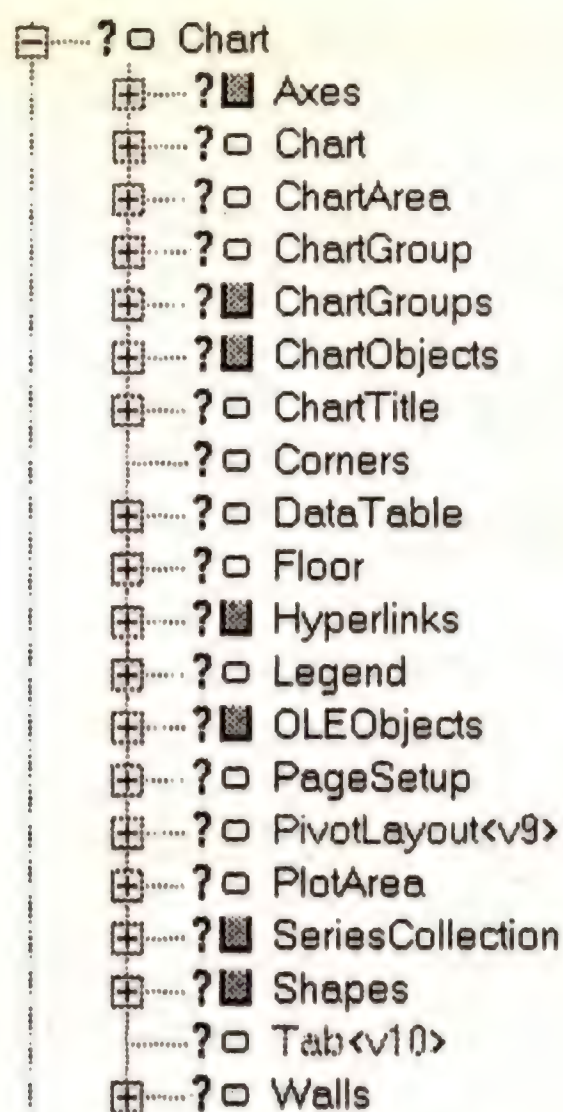


Рис. 21.2. Объект Chart

Соответственно, в этом случае при создании новой диаграммы с помощью мастера четвертый шаг приведет к появлению диалогового окна, показанного на рис. 21.3.

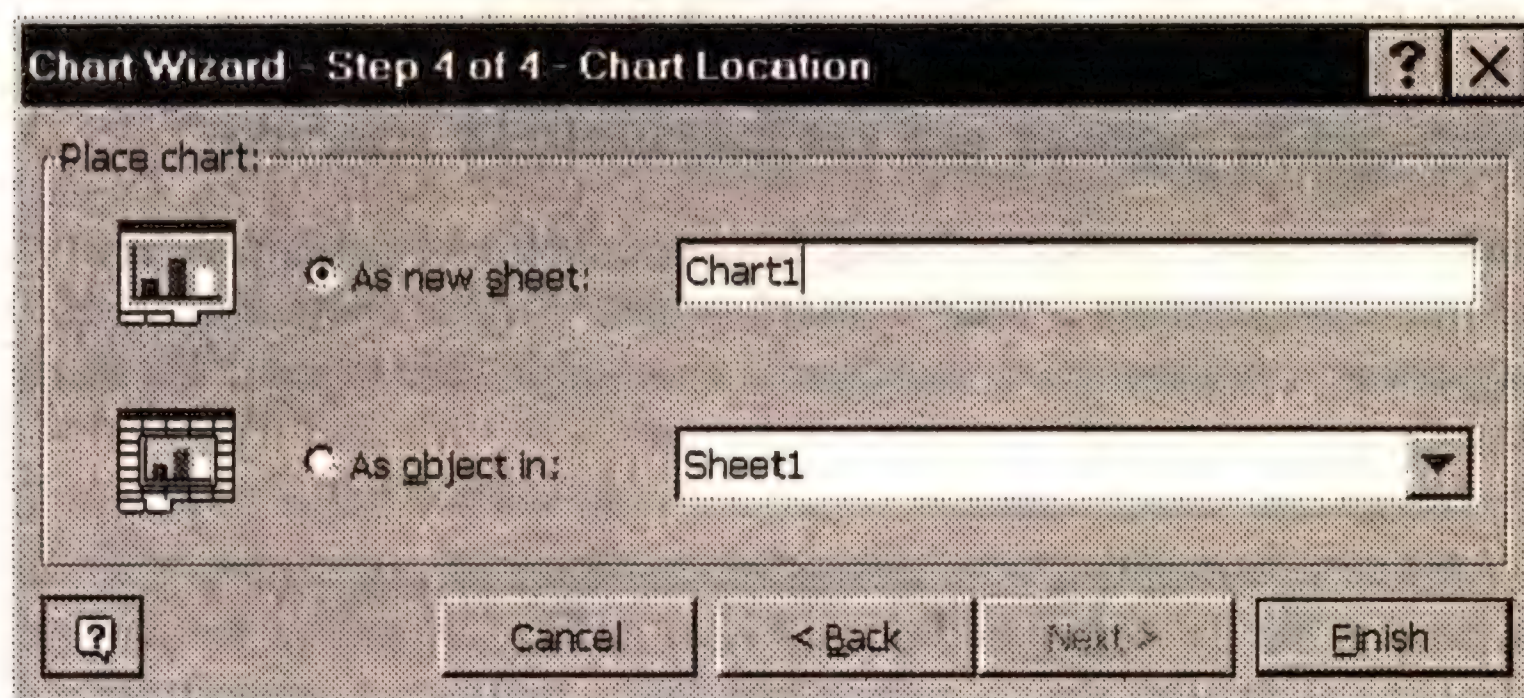


Рис. 21.3. Шаг 4 при работе с мастером диаграмм

Если на четвертом шаге работы с мастером установить переключатель Place chart (Поместить диаграмму на листе) в положение As new sheet (Отдельном), доступ к появляющейся в результате диаграмме можно получить с помощью следующего кода:

```
Dim c as Chart
Set c = ThisWorkbook.Charts("Chart1")
```

С другой стороны, установив переключатель в положение As object in (Имеющемся), мы изменим код доступа к диаграмме:

```
Dim c As Chart
Set c = Worksheets("Sheet1").ChartObjects("Chart 1").Chart
```


Обратите внимание на пробел между словом Chart и числом 1 в имени объекта ChartObject, в то время как в имени объекта Chart пробел отсутствует.

Еще раз подчеркнем, что не существует объекта ChartSheet. Свойство Charts объекта Application возвращает так называемое семейство Sheets, содержащее по одному объекту Chart для каждого листа диаграммы. Но при этом семейство не содержит объектов Chart для встроенных диаграмм.

Создание диаграммы

В предыдущей главе вы узнали о том, что сводные таблицы создаются и добавляются в семейство PivotTables с помощью метода PivotTableWizard. Но создание диаграммы требует иного подхода, так как многое зависит от того, располагается ли диаграмма на отдельном листе или же она является встроенной (и, следовательно, содержится в объекте ChartObject).

В отличие от метода PivotTableWizard метод ChartWizard не создает новую диаграмму, а просто форматирует уже существующий объект. Соответственно, процесс создания диаграммы состоит из трех этапов:

- принятие решения о том, какая именно диаграмма требуется — расположенная на отдельном листе или встроенная;
- создание требуемой диаграммы, как описано в следующем разделе;
- форматирование диаграммы с помощью метода ChartWizard или с помощью индивидуальных свойств и методов объекта Chart.

Создание диаграмм на отдельном листе

Объект Workbook имеет свойство Charts, возвращающее семейство Charts, в которое входят все расположенные в книге диаграммы. Чтобы добавить в это семейство новую диаграмму, следует воспользоваться методом Add семейства Charts.

Вот синтаксис метода Add в данном случае:

```
ChartsObject.Add(Before, After, Count)
```

Как и следовало ожидать, этот метод возвращает объект Chart. Параметр *Before* указывает номер листа, перед которым будет добавлен лист с новой диаграммой, а параметр *After* — номер листа, после которого появится новый лист. Если указать оба этих параметра одновременно, отобразится сообщение об ошибке. Если же не задан ни один из них, новый лист будет включен перед активным листом.

Необязательный параметр *Count* является целым числом, которое соответствует количеству добавляемых листов. По умолчанию ему присвоено значение 1.

Вот пример кода, приводящего к созданию новой диаграммы с именем Sales:

```
Dim ch As Chart  
Set ch = ThisWorkbook.charts.Add()  
ch.Name = "Sales"
```

К сожалению, с помощью метода Add невозможно отформатировать диаграмму. Как уже упоминалось ранее, это осуществляется с помощью различных свойств объекта Chart или с помощью метода ChartWizard, о котором мы поговорим немного позднее.

Создание встроенных диаграмм

Объект `Worksheet` также имеет свойство `ChartObjects`, возвращающее семейство `ChartObjects`, в которое входят все объекты `ChartObjects`, находящиеся на листе. Следуя вышеизложенному, объект `ChartObject` является контейнером объекта `Chart`, представляющего собой встроенную диаграмму.

Семейство `ChartObjects` включает метод `Add`, который используется для создания новых встроенных диаграмм. Вот его синтаксис:

```
ChartsObjectObject.Add(Left, Top, Width, Height).
```

где обязательные параметры *Left* и *Top* указывают на координаты верхнего левого угла диаграммы (в точках) относительно левого верхнего угла ячейки A1. Параметры *Width* и *Height* определяют начальный размер диаграммы (также в точках). Для преобразования дюймов в точки используется метод `InchesToPoints`.

Обратите внимание на то, что метод `Add` возвращает объект `ChartObject`, а не объект `Chart`. Это слегка сбивает с толку, так как метод создает и объект `ChartObject`, и встроенный объект `Chart`. Приведенный в листинге 21.1 код создает новый объект `ChartObject` с именем `ExampleChart` вместе со встроенным объектом `Chart`. Диаграмма размещена таким образом, что ее верхний левый угол отстоит на три столбца от левого края листа и на половину строки от верхнего края. Размер диаграммы составляет 8 столбцов в ширину и 20 строк в высоту.

Листинг 21.1. Создание встроенной диаграммы

```
Sub CreateAChart()  
  
    ' Создание встроенной диаграммы  
  
    Dim co As ChartObject  
    Dim cw As Long, rh As Long  
  
    ' Получение данных для размещения диаграммы  
    cw = Columns(1).Width  
    rh = Rows(1).Height  
  
    ' Размещение диаграммы с применением ширины столбцов и высоты строк в качестве  
    ' единиц измерения  
    Set co = ActiveSheet.ChartObjects.Add(cw * 3, rh * 0.5, cw * 8, rh * 20)  
  
    ' Присваивание имени  
    co.Name = "ChartExample"  
  
    Debug.Print co.Name  
    Debug.Print co.Chart.Name  
  
    ' Задание типа диаграммы  
    co.Chart.ChartType = xlLine  
  
End Sub
```

С помощью операторов `Debug.Print` получаем следующее:

```
ChartExample  
Sheet1 Chart 1
```

Созданная таким способом диаграмма показана на рис. 21.4 (разумеется, она пустая!)

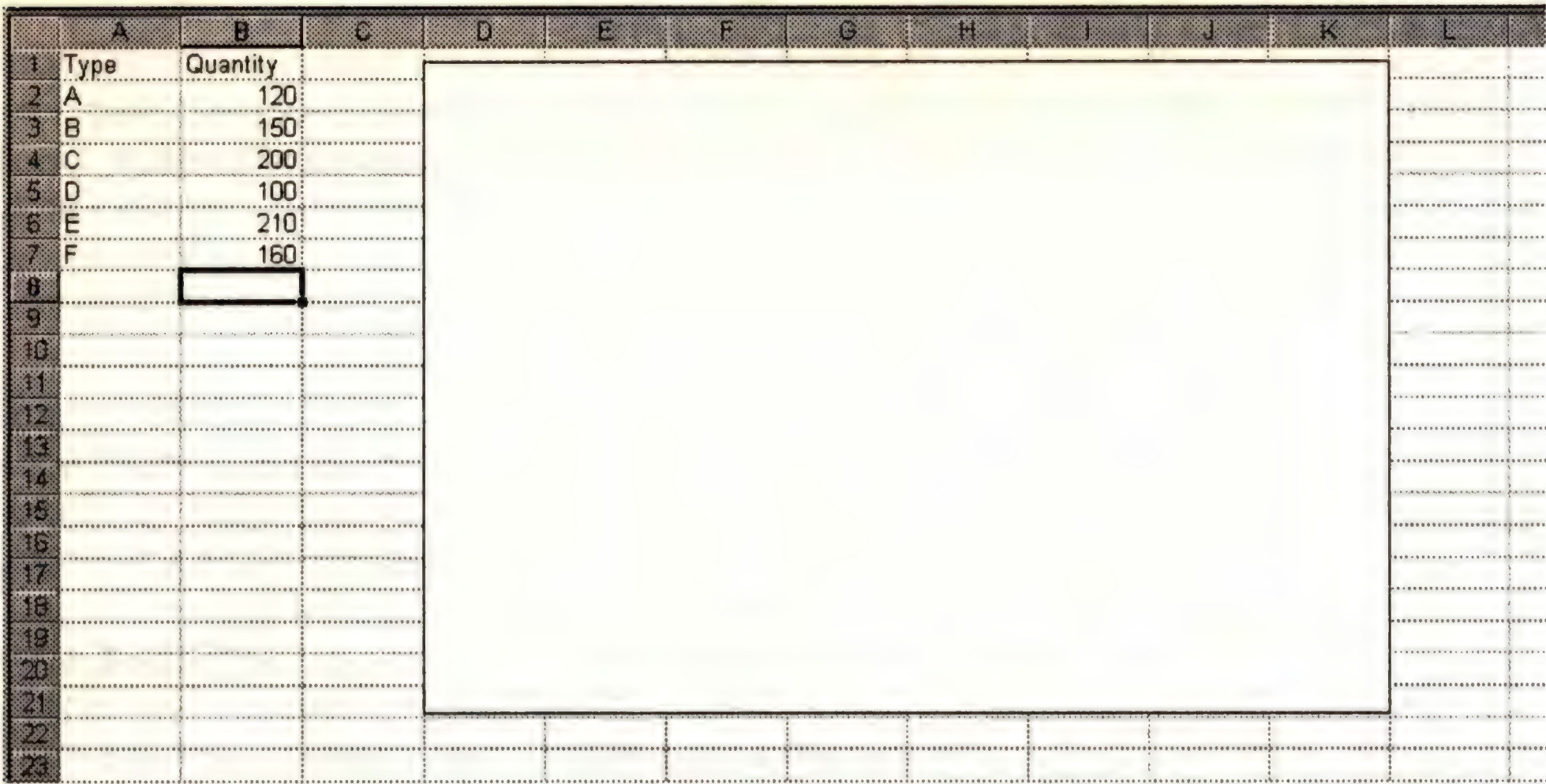


Рис. 21.4. Результат создания встроенной диаграммы

Представленная на рис. 21.4 диаграмма совершенно бесполезна. Для ее изменения потребуется применить различные свойства и методы объекта Chart или же метод ChartWizard.

Хотя в документации это и не упоминается, эксперименты показывают, что свойство Name объекта Chart предназначено только для чтения. И действительно, код:

```
co.Chart.Name = "AChart"
```

приводит к сообщению об ошибке: "Method Name of object _Chart failed."

Обратите внимание: процедура CreateAChart может запускаться на выполнение несколько раз, в результате чего будет получен набор объектов ChartObject с одним и тем же именем! Отсюда можно сделать вывод относительно бесполезности свойства Name как для объектов ChartObject, так и для встроенных объектов Chart. В действительности после двойного запуска процедуры CreateAChart и получения двух диаграмм с именем ExampleChart код:

```
Debug.Print ActiveSheet.ChartObjects(1).Name
Debug.Print ActiveSheet.ChartObjects(2).Name
ActiveSheet.ChartObjects("ChartExample").Left = 600
```

приведет к появлению:

```
ChartExample
ChartExample
```

и переместит одну из диаграмм на новую позицию, указанную свойством Left!

Пример создания диаграммы

Как уже упоминалось, создание осмысленной диаграммы с нуля требует использования свойств и методов объекта Chart. Вскоре вы убедитесь в том, что это очень сложный объект, включающий множество свойств, методов и потомков. Перед началом рассмотрения данных элементов продемонстрируем пример создания диаграммы. В дальнейшем мы не раз будем ссылаться на этот код.

Итак, процедура CreateChart, код которой показан в листинге 21.2, приводит к появлению диаграммы, показанной на рис. 21.5. Может быть, это не самый эффектный пример, но он достаточно хорошо иллюстрирует принцип создания диаграмм.

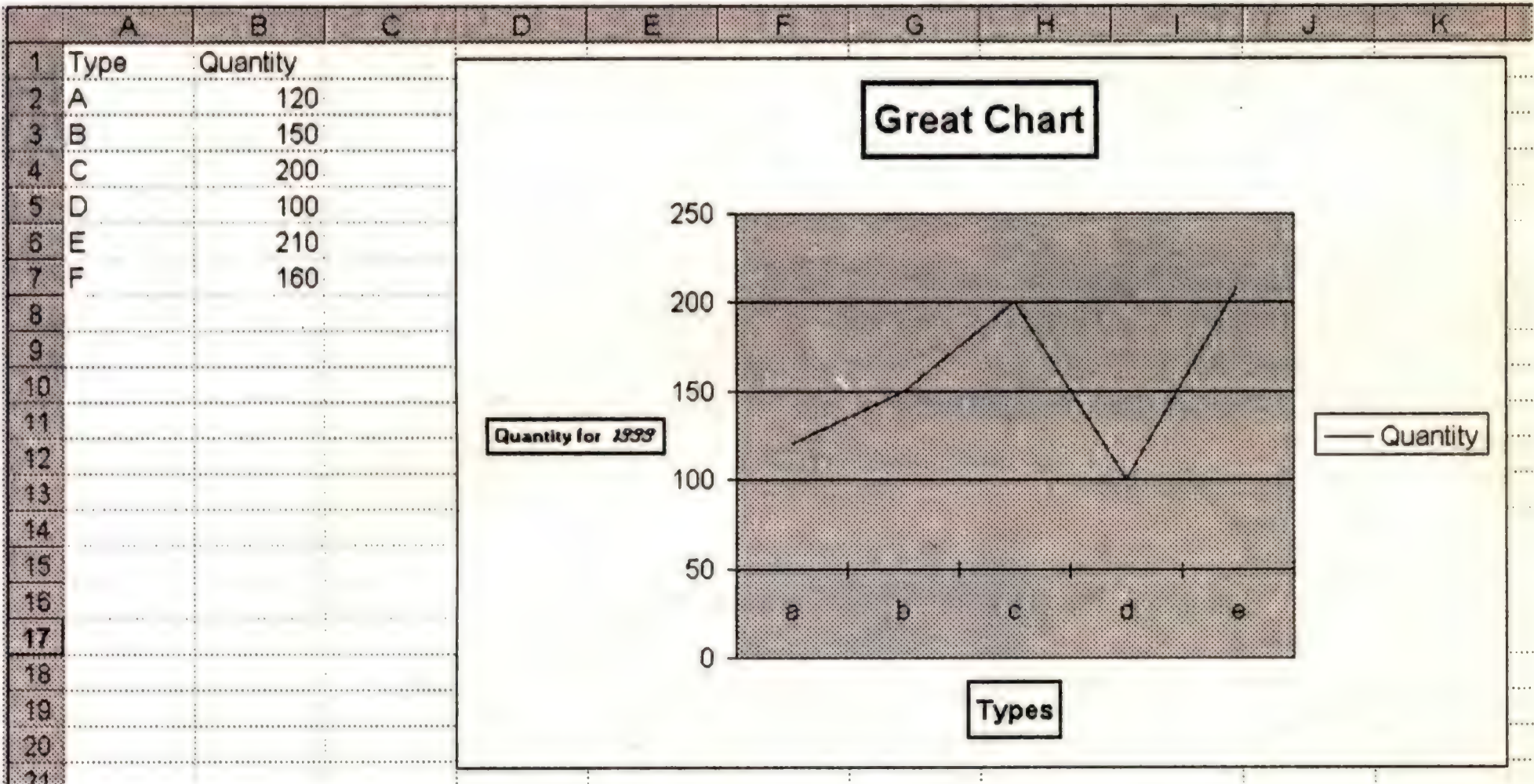


Рис. 21.5. Результат применения процедуры CreateChart

Листинг 21.2. Процедура CreateChart

```
Sub CreateChart()  
  
    ' Создание встроенной диаграммы  
  
    Dim co As ChartObject  
    Dim cw As Long, rh As Long  
  
    ' Получение данных для размещения диаграммы  
    cw = Columns(1).Width  
    rh = Rows(1).Height  
  
    ' Размещение диаграммы с применением ширины столбцов и высоты строк в качестве  
    ' единиц измерения  
    Set co = ActiveSheet.ChartObjects.Add(cw * 3, rh * 0.5, cw * 8, rh * 20)  
  
    ' Присваивание имени диаграмме  
    co.Name = "ChartExample"  
  
    ' Определение типа диаграммы  
    co.Chart.ChartType = xlLine  
  
    ' Добавление рядов данных  
    co.Chart.SeriesCollection.Add  
        Source:=ActiveSheet.Range("A1:B6"),  
        Rowcol:=xlColumns, SeriesLabels:=True, _  
        CategoryLabels:=True  
  
    ' Добавление осей
```

продолжение ➤

Листинг 21.2 (продолжение)

```

' (В действительности это задано по умолчанию,
' но в данный пример был добавлен соответствующий процесс для иллюстрации)
With co.Chart
    .HasAxis(xlCategory, xlPrimary) = True
    .HasAxis(xlCategory, xlSecondary) = False
    .HasAxis(xlValue, xlPrimary) = True
    .HasAxis(xlValue, xlSecondary) = False
End With

' Форматирование подписей к осям
With co.Chart.Axes(xlCategory)
    .HasTitle = True
    .AxisTitle.Caption = "Types"
    .AxisTitle.Border.Weight = xlMedium
End With
With co.Chart.Axes(xlValue)
    .HasTitle = True
    With .AxisTitle
        .Caption = "Quantity for 1999"
        .Font.Size = 6
        .Orientation = xlHorizontal
        .Characters(14, 4).Font.Italic = True
        .Border.Weight = xlMedium
    End With
End With

' Изменение названия категории (Types) с применением букв нижнего
' регистра
' (На листе они появляются отображенными буквами верхнего регистра)
co.Chart.Axes(xlCategory).CategoryNames = _
    Array("a", "b", "c", "d", "e")

' Задание точки пересечения осей значений на отметке 50
co.Chart.Axes(xlValue).CrossesAt = 50

' Формирование только горизонтальных линий сетки
co.Chart.Axes(xlValue).HasMajorGridlines = True
co.Chart.Axes(xlCategory).HasMajorGridlines = False

' Временные метки на оси категорий
co.Chart.Axes(xlCategory).MajorTickMark = xlTickMarkCross

' Перемещение подписей к временным меткам под диаграмму
co.Chart.Axes(xlCategory).TickLabelPosition = _
    xlTickLabelPositionNextToAxis

' Заполнение области диаграммы белым цветом
co.Chart.ChartArea.Interior.Color = RGB(255, 255, 255)

' Заполнение области графика серым цветом
co.Chart.PlotArea.Interior.ColorIndex = 15

' Форматирование заголовка диаграммы
With co.Chart.ChartTitle
    .Caption = "Great Chart"
    .Font.Size = 14

```



```
.Font.Bold = True
.Border.Weight = xlThick
End With

End Sub
```

Порядок расположения по оси Z объектов ChartObject

Перед тем как перейти к рассмотрению основных свойств, методов и потомков объекта Chart, упомянем один важный момент, а именно каким образом осуществляется контроль порядка отображения встроенных диаграмм при их перекрывании. Каждый объект ChartObject имеет так называемый z-порядок, указывающий относительное положение объекта по отношению к воображаемой оси z, как бы выходящей из монитора под углом 90°, как показано на рис. 21.6.

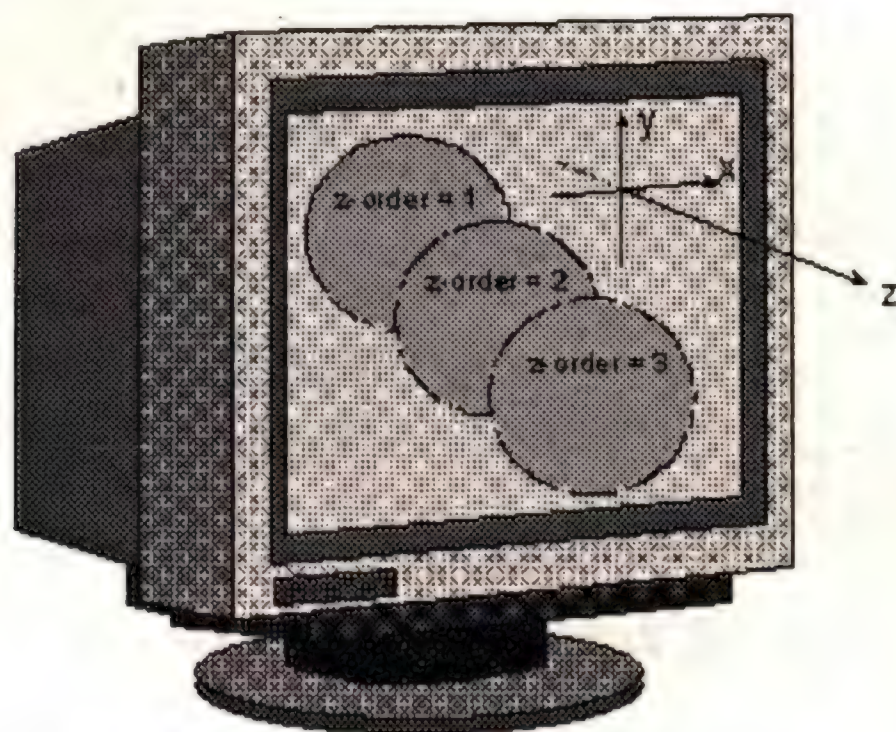


Рис. 21.6. Иллюстрация z-порядка

Объект ChartObject имеет предназначенное только для чтения свойство ZOrder, которое возвращает z-порядок этого объекта. Он также включает методы BringToFront и SendToBack, позволяющие изменять z-порядок.

Типы диаграмм

Каждая диаграмма Excel принадлежит к стандартному или к нестандартному типу. Кроме того, существуют два дополнительных нестандартных типа: встроенный и определенный пользователем. Для доступа к списку типов диаграмм используется диалоговое окно Chart Type (Тип диаграммы), показанное на рис. 21.7. Для его вызова достаточно щелкнуть правой кнопкой мыши на диаграмме и в контекстном меню выбрать команду Chart Type (Тип диаграммы).

Свойство ChartType

Редактируемое свойство ChartType может принимать значения констант xlChartType, перечисленных в табл. 21.1.

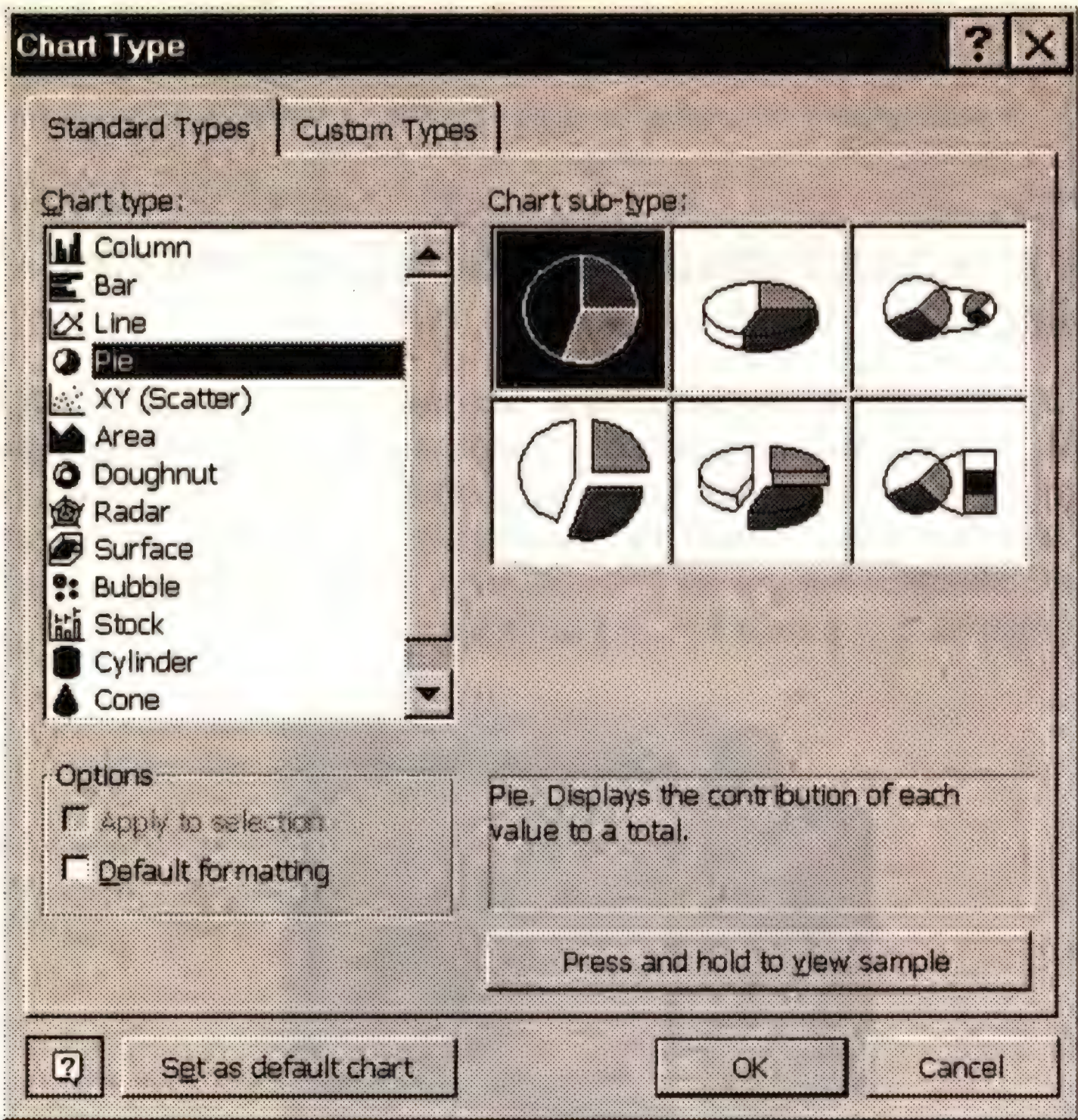


Рис. 21.7. Диалоговое окно Chart Type

Таблица 21.1. Константы ChartType

Тип	Описание	Значение
Area (С Областями)	3D Area	xl3DArea(-4098)
	3D Stacked Area	xl3DAreaStacked(78)
	3D 100% Stacked Area	xl3DAreaStacked100(79)
	Area	xlArea(1)
	Stacked Area	xlAreaStacked(76)
	100% Stacked Area	xlAreaStacked100(77)
Bar (Линейчатая)	3D Clustered Bar	xl3DBarClustered(60)
	3D Stacked Bar	xl3DBarStacked(61)
	3D 100% Stacked Bar	xl3DBarStacked100(62)
	Clustered Bar	xlBarClustered(57)
	Stacked Bar	xlBarStacked(58)
	100% Stacked Bar	xlBarStacked100(59)
Bubble (Пузырьковая)	Bubble	xlBubble(15)
	Bubble with 3D effects	xlBubble3DEffect(87)
Column (Гистограмма)	3D Column	xl3DColumn(-4100)
	3D Clustered Column	xl3DColumnClustered(54)
	3D Stacked Column	xl3DColumnStacked(55)

Тип	Описание	Значение
Column (Гистограмма)	3D 100% Stacked Column	xl3DColumnStacked100(56)
	Clustered Column	xlColumnClustered(51)
	Stacked Column	xlColumnStacked(52)
	100% Stacked Column	xlColumnStacked100(53)
Cone (Коническая)	Clustered Cone Bar	xlConeBarClustered(102)
	Stacked Cone Bar	xlConeBarStacked(103)
	100% Stacked Cone Bar	xlConeBarStacked100(104)
	3D Cone Column	xlConeCol(105)
	Clustered Cone Column	xlConeColClustered(99)
	Stacked Cone Column	xlConeColStacked(100)
	100% Stacked Cone Column	xlConeColStacked100(101)
Cylinder (Цилиндрическая)	Clustered Cylinder Bar	xlCylinderBarClustered(95)
	Stacked Cylinder Bar	xlCylinderBarStacked(96)
	100% Stacked Cylinder Bar	xlCylinderBarStacked100(97)
	3D Cylinder Column	xlCylinderCol(98)
	Clustered Cylinder Column	xlCylinderColClustered(92)
	Stacked Cylinder Column	xlCylinderColStacked(93)
	100% Stacked Cylinder Column	xlCylinderColStacked100(94)
Doughnut (Кольцевая)	Doughnut	xlDoughnut(-4120)
	Exploded Doughnut	xlDoughnutExploded(80)
Line (График)	3D Line	xl3DLine(-4101)
	Line	xlLine(4)
	Line with Markers	xlLineMarkers(65)
	Stacked Line with Markers	xlLineMarkersStacked(66)
	100% Stacked Line with Markers	xlLineMarkersStacked100(67)
	Stacked Line	xlLineStacked(63)
Pie (Круговая)	100% Stacked Line	xlLineStacked100(64)
	3D Pie	xl3DPie(-4102)
	Exploded 3D Pie	xl3DPieExploded(70)
	Bar of Pie	xlBarOfPie(71)
	Pie	xlPie(5)
	Exploded Pie	xlPieExploded(69)
Pyramid (Пирамидальная)	Pie of Pie	xlPieOfPie(68)
	Clustered Pyramid Bar	xlPyramidBarClustered(109)
	Stacked Pyramid Bar	xlPyramidBarStacked(110)
	100% Stacked Pyramid Bar	xlPyramidBarStacked100(111)
	3D Pyramid Column	xlPyramidCol(112)
	Clustered Pyramid Column	xlPyramidColClustered(106)
	Stacked Pyramid Column	xlPyramidColStacked(107)
Radar (Лепестковая)	100% Stacked Pyramid Column	xlPyramidColStacked100(108)
	Radar	xlRadar(-4151)
	Filled Radar	xlRadarFilled(82)
	Radar with Data Markers	xlRadarMarkers(81)

Таблица 21.1 (продолжение)

Тип	Описание	Значение
Stock Quotes (Биржевая)	High-Low-Close	xlStockHLC(88)
	Open-High-Low-Close	xlStockOHLC(89)
	Volume-High-Low-Close	xlStockVHLC(90)
	Volume-Open-High-Low-Close	xlStockVOHLC(91)
Surface (Поверхность)	3D Surface	xlSurface(83)
	Surface (Top View)	xlSurfaceTopView(85)
	Surface (Top View) wireframe)	xlSurfaceTopViewWireframe(86)
	3D Surface (wireframe)	xlSurfaceWireframe(84)
XY Scatter (Точечная)	Scatter	xlXYScatter(-4169)
	Scatter with Lines	xlXYScatterLines(74)
	Scatter with Lines and No Data Markers	xlXYScatterLinesNoMarkers(75)
	Scatter with Smoothed Lines	xlXYScatterSmooth(72)
	Scatter with Smoothed Lines and No Data Markers	xlXYScatterSmoothNoMarkers(73)

Для получения подробного описания каждой диаграммы на русском языке откройте диалоговое окно Chart Type (Тип диаграммы), выделите в списке пункт Chart Type (Тип диаграммы), затем — в списке Chart sub-type (Вид) диаграмму нужного вида, и в нижнем правом углу появится соответствующее описание.

В листинге 21.3 (см. с. 417) приводится код макроса, осуществляющего прокрутку типов диаграмм, перечисленных в табл. 21.1. Это позволяет быстро определить вид диаграммы, который наилучшим образом подходит для ваших целей.

Иногда свойство ChartType возвращает значение, не указанное в табл. 21.1. Например, код:

```
MsgBox ActiveChart.ChartType
```

возвращает -4111 в применении к диаграмме пользовательского типа. В объектной модели Excel это значение фигурирует только один раз. Оно является частью перечисления Constants и имеет символьное имя xlCombination. (Так как в документации автору не удалось найти упоминания об этой особенности свойства ChartType, нельзя однозначно утверждать о том, что это единственное значение данного свойства, не содержащееся в табл. 21.1).

Исходя из того, что на основе набора данных можно построить диаграмму любого типа, приходим к выводу о том, что объект Series также обладает свойством ChartType. Соответственно, если диаграмма построена на основе двух наборов данных, каждый из наборов может формировать диаграмму собственного типа. Например, один ряд данных формирует диаграмму Line (График), а второй — диаграмму Column (Гистограмма). Подробная информация об объекте Series будет дана чуть позднее.

Метод ApplyCustomType

Несмотря на свое название, метод ApplyCustomType может работать с диаграммами, относящимися к стандартному и нестандартному типам. Вот его синтаксис:

```
ChartObject.ApplyCustomType(ChartType, TypeName)
```


Параметр *ChartType* принимает значения констант как из табл. 21.1, так и из следующего перечисления:

```
Enum XlChartGallery
    xlBuiltIn = 21
    xlUserDefined = 22
    xlAnyGallery = 23
End Enum
```

В справочной документации Excel автору не встречался термин ChartGallery. Однако в объектной модели Word существует термин ListGallery, который относится к объектам, соответствующим трем вкладкам диалогового окна Bullets and Numbering. Экстраполируя это на Excel, можно предположить, что термин ChartGallery относится к двум вкладкам диалогового окна Chart Type (Тип диаграммы), показанного на рис. 21.8.

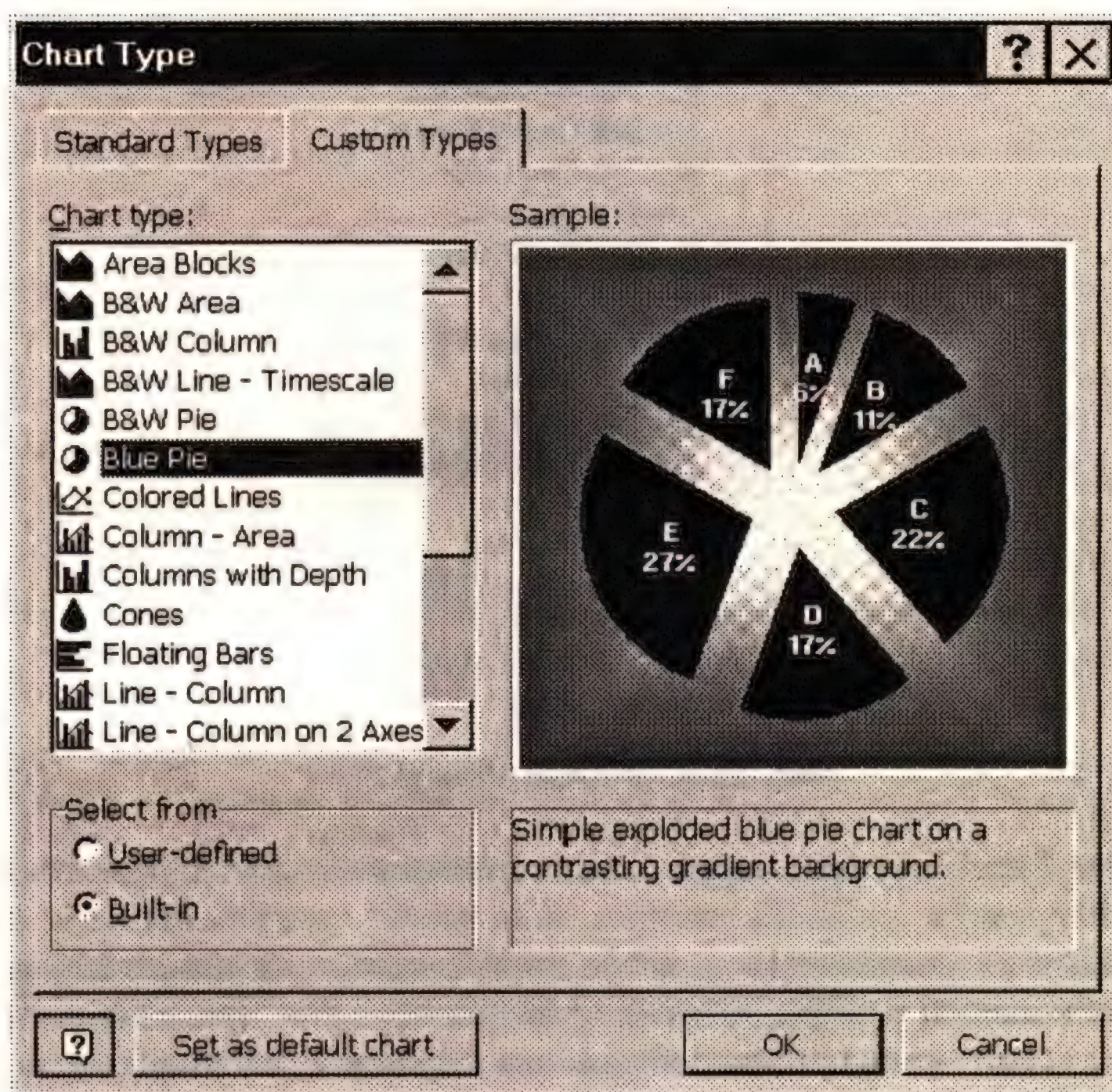


Рис. 21.8. Иллюстрация метода ApplyCustomType

Если параметр *ChartType* принимает значения одной из констант перечисления XlChartGallery, то необязательный параметр *TypeName* указывает название типа диаграммы.

Например, следующий код:

```
ActiveChart.ApplyCustomType _
    ChartType:=xlBuiltIn, TypeName:="Blue Pie"
```

применяет встроенную диаграмму Blue Pie (Голубая круговая). Это эквивалентно выделению варианта Blue Pie (Голубая круговая) в диалоговом окне, показанном на рис. 21.8.

В качестве следующего примера рассмотрим код, устанавливающий дополнительный тип диаграммы с именем DefaultXY:

```
ActiveChart.ApplyCustomType
ChartType:=xlUserDefined, TypeName:="DefaultXY"
```

Потомки объекта Chart

На рис. 21.9 показаны потомки объекта Chart.

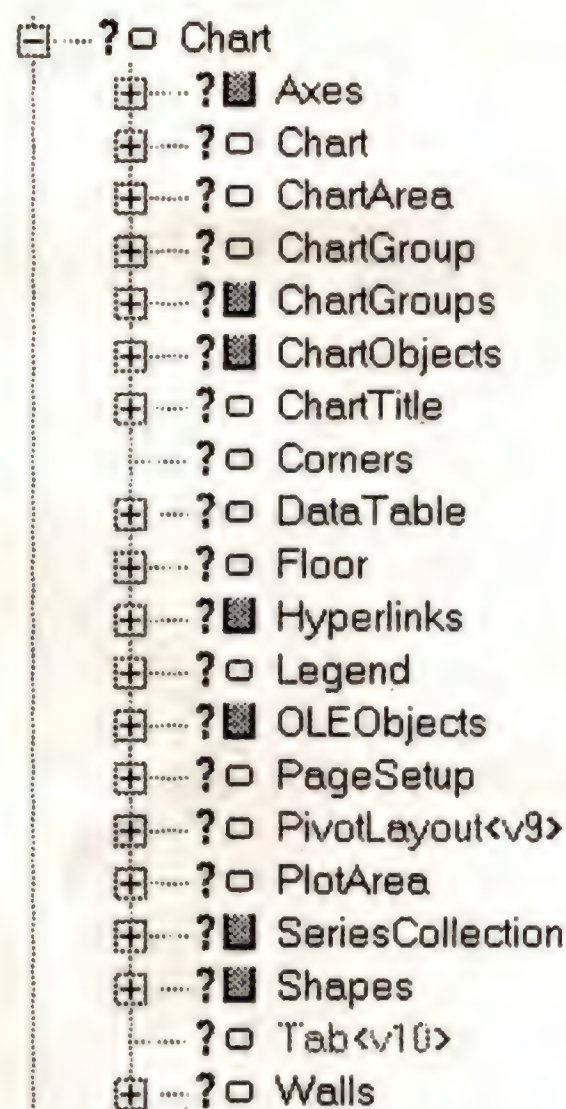


Рис. 21.9. Потомки объекта Chart

Диаграмма Excel состоит из нескольких компонентов: оси, область диаграммы, заголовок диаграммы, таблица данных, основание диаграммы (для трехмерных диаграмм), область, фактически занятая диаграммой, и один или несколько рядов данных (с подписями и значениями). Эти компоненты представляют собой потомки объекта Chart, к изучению которых мы перейдем в этом разделе.

Семейство Axes

На рис. 21.10 показана часть объектной модели Excel, связанная с осями диаграммы.

Объект Chart имеет семейство Axes, включающее объекты Axis для каждой оси диаграммы. Метод Axes возвращает одну ось или целое семейство Axes для рассматриваемой диаграммы. Чтобы вернуть семейство Axes диаграммы, достаточно написать:

```
ChartObject.Axes
```

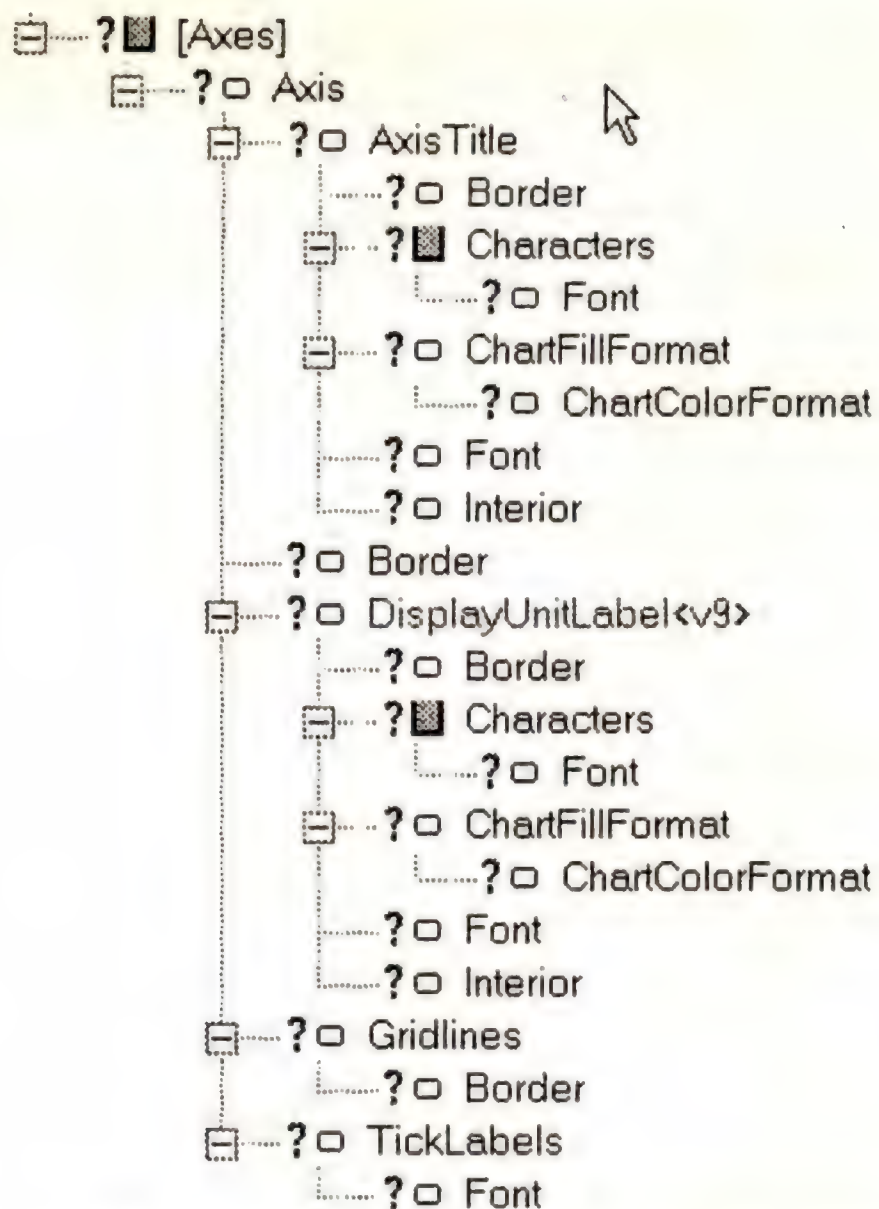



Рис. 21.10. Объекты, связанные с осями диаграммы

Если же требуется вернуть определенный объект Axis, используется следующий код:

```
ChartObject.Axes(Type, AxisGroup)
```

В данном случае необязательный параметр *Type* указывает, какая ось должна быть возвращена. Он может принимать значения констант из следующего перечисления:

```
Enum xlAxisType
    xlCategory = 1      ' Ось категорий
    xlValue = 2         ' Ось значений
    xlSeriesAxis = 3    ' Ось рядов данных
End Enum
```

Значение *xlSeriesAxis* используется только в трехмерных диаграммах.

Необязательный параметр *AxisGroup* определяет группу осей. Он может принимать значения констант из следующего перечисления:

```
Enum xlAxisGroup
    xlPrimary = 1       ' По умолчанию
    xlSecondary = 2
End Enum
```

Вторичной осью называется вторая вертикальная или горизонтальная ось, на которой указываются вторые ряды значений (в случае вертикальной оси) или ряды категорий (в случае горизонтальной оси) (см. рис. 21.1). При пропуске этого аргумента используется первичная группа осей. Трехмерные диаграммы имеют только одну группу осей.

Объект Chart обладает редактируемым свойством HasAxis, которое определяет, будут ли на диаграмме показаны оси различных типов. Следует отметить, что свойство не будет задействовано при отсутствии соответствующих рядов данных.

Например, при создании диаграммы с нуля добавляется код, формирующий ось категорий до момента появления данных, которые будут располагаться вдоль этой оси. Но этого не следует делать!

Вернемся к процедуре `CreateChart`, упомянутой чуть ранее. Если изменить код, связанный с данными, и код, связанный с осью, написав вместо фрагмента кода:

```
' Добавление рядов данных
co.Chart.SeriesCollection.Add
    Source:=ActiveSheet.Range("A1:B6"), _
    Rowcol:=xlColumns, SeriesLabels:=True, _
    Categorylabels:=True

' Добавление оси
With co.Chart
    .HasAxis(xlCategory, xlPrimary) = True
    .HasAxis(xlCategory, xlSecondary) = False
    .HasAxis(xlValue, xlPrimary) = True
    .HasAxis(xlValue, xlSecondary) = False
End With
```

код:

```
' Добавление оси
With co.Chart
    .HasAxis(xlCategory, xlPrimary) = True
    .HasAxis(xlCategory, xlSecondary) = False
    .HasAxis(xlValue, xlPrimary) = True
    .HasAxis(xlValue, xlSecondary) = False
End With
```

```
' Добавление ряда данных
co.Chart.SeriesCollection.Add
    Source:=ActiveSheet.Range("A1:B6"), _
    Rowcol:=xlColumns, SeriesLabels:=True, _
    Categorylabels:=True.
```

отобразится совершенно бесполезное сообщение об ошибке: `Method 'HasAxis' of object '_Chart' has failed`. К сожалению, эта проблема совершенно не упоминается в справочной документации Excel.

Вот синтаксис свойства `HasAxis`:

```
ChartObject.HasAxis(Index1, Index2).
```

где параметры, несмотря на общие имена, непосредственно соответствуют параметрам *Type* и *AxisGroup* метода *Axes*. Ниже приводится пример кода, отображающего первичную ось категорий, а также первичную и вторичную оси значений для активной диаграммы:

```
With ActiveChart
    .HasAxis(xlCategory, xlPrimary) = True
    .HasAxis(xlCategory, xlSecondary) = False
    .HasAxis(xlValue, xlPrimary) = True
    .HasAxis(xlValue, xlSecondary) = True
End With
```

Объект Axis

В табл. 21.2 перечислены свойства и методы объекта *Axis*. Смысл большинства из них понятен по названиям, поэтому они будут рассмотрены достаточно кратко.

Большинство этих свойств и методов соответствуют бесчисленным флажкам и спискам пяти вкладок диалогового окна Format Axis (Формат оси).

Таблица 21.2. Свойства и методы объекта Axis

Application	HasDisplayUnitLabel<v9>	MinorTickMark
AxisBetweenCategories	HasMajorGridlines	MinorUnit
AxisGroup	HasMinorGridlines	MinorUnitIsAuto
AxisTitle	HasTitle	MinorUnitScale
BaseUnit	Height	Parent
BaseUnitIsAuto	Left	ReversePlotOrder
Border	MajorGridlines	ScaleType
CategoryNames	MajorTickMark	Select
CategoryType	MajorUnit	TickLabelPosition
Creator	MajorUnitIsAuto	TickLabels
Crosses	MajorUnitScale	TickLabelSpacing
CrossesAt	MaximumScale	TickMarkSpacing
Delete	MaximumScaleIsAuto	Top
DisplayUnit<v9>	MinimumScale	Type
DisplayUnitCustom<v9>	MinimumScaleIsAuto	Width
DisplayUnitLabel<v9>	MinorGridlines	

Свойство AxisGroup

Предназначенное только для чтения свойство AxisGroup возвращает группу, к которой принадлежит определенная ось. Оно может принимать значение xlPrimary (= 1) или xlSecondary (= 2).

Заголовки осей и их форматирование

Свойство AxisTitle возвращает объект AxisTitle, представляющий собой заголовок определенной оси.

Свойство HasTitle (чтение/запись, тип Boolean)

Перед форматированием заголовка оси требуется указать, что у оси присутствует заголовок. Для этого применяется свойство HasTitle:

```
AxisObject.HasTitle = True
```

Перечислим наиболее важные свойства объекта AxisTitle:

- если свойство AutoScaleFont имеет заданное по умолчанию значение True, при изменении размеров диаграммы происходит автоматическое пропорциональное изменение размера шрифта, которым написан заголовок;
- свойство Border возвращает объект Border, с помощью которого можно указать цвет, стиль линии и ширину границы заголовка оси;
- свойство Caption определяет текст подписи;
- свойство Characters возвращает объект Characters, представляющий собой совокупность символов в подписи (с помощью этого свойства можно отформатировать часть текста подписи);

- свойство Fill возвращает объект ChartFillFormat, задающий заливку фона в заголовке оси;
- свойство Font возвращает объект Font, который определяет шрифт подписей;
- свойства HorizontalAlignment и VerticalAlignment выравнивают текст заголовка по горизонтали и вертикали;
- свойство Interior возвращает объект Interior, используемый для форматирования внутренней части заголовка оси;
- свойство NumberFormat задает код числового формата для подписей, причем если подписи отформатированы различными способами, это свойство возвращает значение Null (так как в данном случае используются те же самые форматы кода, что и при форматировании ячеек листа, для их получения можно воспользоваться подпрограммой формирования, записи и сохранения клавиатурной макрокоманды);
- свойство Orientation возвращает или задает ориентацию заголовка оси, причем это свойство принимает значения констант из следующего перечисления:

```
Enum xlOrientation
    xlUpward = -4171
    xlDownward = -4170
    xlVertical = -4166
    xlHorizontal = -4128
End Enum
```

Обратите внимание на объект Characters, который представляет собой часть *непрерывного* текста внутри текстовой строки. Он возвращает свойство Characters. Впрочем, это свойство также применимо к объектам ChartTitle и Range.

Вот синтаксис свойства Characters:

```
AxisTitleObject.Characters(start, length),
```

где параметр *start* определяет положение первого символа рассматриваемой части текста, а параметр *length* — количество символов во фрагменте текста. Если параметр *start* не указан пользователем, ему автоматически присваивается значение 1. Если же пропущен аргумент *length*, будет рассматриваться строка до самого конца.

Для иллюстрации рассмотрим код, который создает заголовок для первичной оси значений, а потом выделяет курсивом слово миллиард. Обратите внимание на то, что сначала задается свойство HasTitle.

```
With ActiveChart.Axes(xlValue, xlPrimary)
    .HasTitle = True
    .AxisTitle.Text = "Это миллиарды"
    .AxisTitle.Characters(5, 9).Font.Italic = True
End With
```

Если бы потребовалось выделить курсивом весь заголовок, можно было бы просто воспользоваться свойством Font объекта AxisTitle:

```
ActiveChart.Axes(xlValue, xlPrimary). _
    AxisTitle.Font.Italic = True
```

Свойство Border и объект Border

Свойство Border возвращает объект Border, представляющий собой границу диаграммы. С помощью этого объекта можно задать цвет, стиль линии и толщину границы объекта, например заголовка оси.

Объект Border лишен методов. Его основными свойствами являются Color, ColorIndex, LineStyle и Weight.

Свойство Color может принимать любое значение RGB. Например, следующий код назначает первичным осям синий цвет:

```
ActiveChart.Axes(xlCategory, xlPrimary).  
MajorGridlines.Border.Color = RGB(0, 0, 255)
```

Свойство ColorIndex используется для задания цвета с помощью палитры цветов. Подробную информацию о значениях этого свойства можно найти в справочной документации.

Свойство LineStyle может принимать значения констант из следующего перечисления:

```
Enum xlLineStyle  
xlLineStyleNone = -4142  
xlDouble = -4119  
xlDot = -4118  
xlDash = -4115  
xlContinuous = 1  
xlDashDot = 4  
xlDashDotDot = 5  
xlSlantDashDot = 13  
End Enum
```

Свойство Weight принимает значения констант из следующего перечисления:

```
Enum xlBorderWeight  
xlMedium = -4138  
xlHairline = 1  
xlThin = 2  
xlThick = 4  
End Enum
```

Дальнейшая иллюстрация процедуры форматирования заголовков осей будет связана с кодом рассмотренной ранее процедуры CreateChart:

```
' Форматирование осей  
With co.Chart.Axes(xlCategory)  
    .HasTitle = True  
    .AxisTitle.Caption = "Types"  
    .AxisTitle.Border.Weight = xlMedium  
End With  
  
With co.Chart.Axes(xlValue)  
    .HasTitle = True  
    With .AxisTitle  
        .Caption = "Quantity for 1999"  
        .Font.Size = 6  
        .Orientation = xlHorizontal  
        .Characters(14, 4).Font.Italic = True  
        .Border.Weight = xlMedium  
    End With  
End With
```

Свойство CategoryNames

Свойство CategoryNames возвращает (или задает) названия категорий осей. Его можно сопоставить как с массивом, так и с объектом Range, содержащим имена категорий.

Например, код:

```
ActiveChart.Axes(xlCategory, xlSecondary).
    CategoryNames = Array("Один", "Два", "Три", "Четыре", "Пять", "Шесть")
```

изменяет метки на верхней горизонтальной оси, присваивая им значения Один, "Два", "Три" и т. д.

Свойства CategoryType и BaseUnit

Свойство CategoryType возвращает или задает тип оси категорий (к осям значений не применимо). Оно может принимать значения констант из следующего перечисления:

```
Enum xlCategoryType
    xlAutomaticScale = -4105    ' Автоматический масштаб
    xlCategoryScale = 2        ' Масштаб категорий
    xlTimeScale = 3            ' Временной масштаб
End Enum
```

По умолчанию это свойство имеет значение xlAutomaticScale.

Если свойство CategoryType для оси имеет значение xlTimeScale, свойство BaseUnit возвращает или задает единицы измерения на выбранной оси категорий. Оно может принимать значения констант из следующего перечисления:

```
Enum xlTimeUnit
    xlDays = 0        ' Дни
    xlMonths = 1      ' Месяцы
    xlYears = 2       ' Годы
End Enum
```

Имейте в виду, что значение свойства BaseUnit проявляется только в том случае, если свойству CategoryType присвоено значение xlTimeScale.

Свойства Crosses и CrossesAt

Свойство Crosses возвращает или определяет точку выбранной оси, в которой она пересекается с другой осью. При этом мы предполагаем, что обе оси принадлежат к одной и той же группе. То есть если указанная ось является первичной, то же самое можно сказать о пересекающей ее оси. Например, код:

```
ActiveChart.Axes(xlValue, xlPrimary).Crosses = xlAxisCrossesMaximum
```

заставляет первичную ось категорий пересекать первичную ось значений в точке максимального значения.

Свойство Crosses может принимать значения констант из следующего перечисления:

```
Enum xlAxisCrosses
    xlAxisCrossesCustom = -4114
    xlAxisCrossesAutomatic = -4105
    xlAxisCrossesMaximum = 2
    xlAxisCrossesMinimum = 4
End Enum
```

Если присвоить свойству Crosses значение xlAxisCrossesAutomatic, точка пересечения будет определяться самим приложением. Присваивание значения xlMinimum приводит к тому, что ось значений пересекается в точке минимального значения, а ось категорий — в точке крайней левой категории. Соответственно, присвоение

значения `xlMaximum` приводит к тому, что ось значений пересекается в точке максимального значения, а ось категорий в точке крайней правой категории. К сожалению, это свойство не всегда работает корректно.

Ну и наконец, значение `xlAxisCrossesCustom` свойства `Crosses` применяется только к оси значений. В этом случае точку пересечения с осью категорий задает свойство `CrossesAt`. Имейте в виду, что определение значения свойства `CrossesAt` приводит к автоматическому присвоению свойству `Crosses` значения `xlAxisCrossesCustom`.

Например, в процедуре `CreateChart` код:
`co.Chart.Axes(xlValue).CrossesAt = 50`
приводил к тому, что ось категорий пересекала ось значений в точке со значением 50. Напоследок заметим, что все эти свойства не применимы к трехмерным диаграммам.

Отображение значений

В версии Excel 9 появились свойства, регулирующие отображение значений `HasDisplayUnitLabel`, `DisplayUnitLabel`, `DisplayUnit` и `DisplayUnitCustom`, а также объект `DisplayUnitLabel`. Эти свойства полезны, когда требуется разработать легенду единиц измерения для осей диаграммы, например при использовании очень больших единиц. Рассмотрим набор меток в форме 1000000, 2000000, 3000000 и т. д. В этом случае можно создать легенду со словом «миллион» и изменить с использованием формата 1, 2, 3, ...

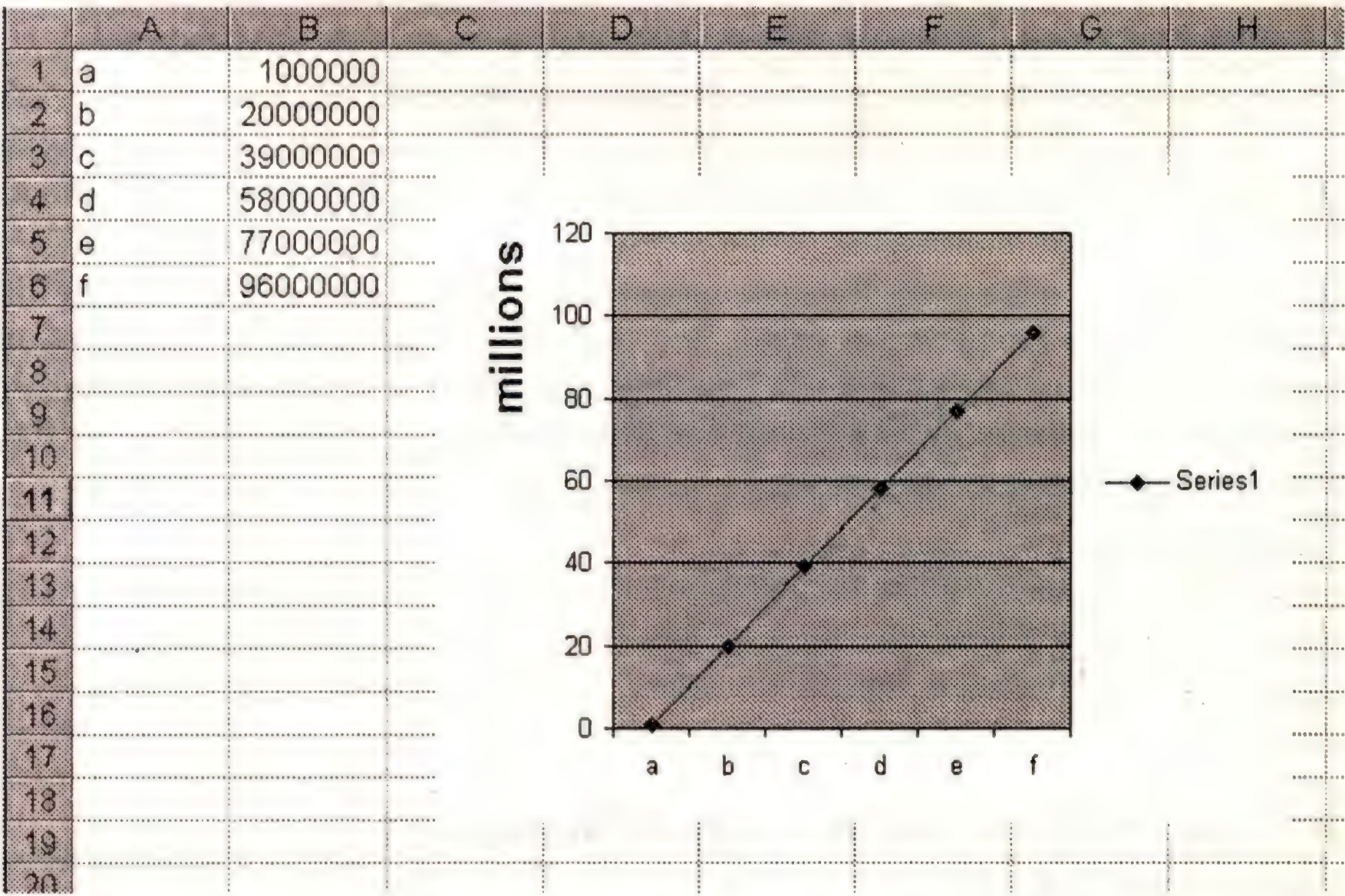


Рис. 21.11. Отображение значений с легендой

В качестве примера рассмотрим код:
`Sub DisplayUnitLabel()
With ActiveChart.Axes(xlValue)`


```

        .DisplayUnit = xlMillions
        .HasDisplayUnitLabel = True
    With .DisplayUnitLabel
        .Caption = "millions"
        .Font.Size = 14
    End With
End With
End Sub.

```

создающий диаграмму, изображенную на рис. 21.11. Обратите внимание, что ось значений имеет пометку millions (миллионы), как и указано в легенде.

Свойства, связанные с линиями сетки, и объект Gridline

Объект Axis имеет ряд свойств, связанных с линиями сетки:

- В результате присваивания редактируемому свойству HasMajorGridlines значения True начинают отображаться основные линии сетки для выбранной оси. Соответственно, значение False приводит к сокрытию данных линий. Это свойство применимо только к первичным осям.
- Благодаря присваиванию редактируемому свойству HasMinorGridlines значения True становятся видимыми промежуточные линии сетки. Соответственно, значение False скрывает эти линии. Это свойство применимо только к первичным осям.
- Предназначенное только для чтения свойство MajorGridlines возвращает объект Gridlines, который представляет собой основные линии сетки для указанной пользователем оси. Это свойство применимо только к первичным осям.
- Предназначенное только для чтения свойство MinorGridlines возвращает объект Gridlines, который представляет собой промежуточные линии сетки для указанной пользователем оси. Это свойство применимо только по отношению к первичным осям.

Имейте в виду, что объект Gridlines не является семейством; то есть объект Gridline в объектной модели Excel отсутствует. Все дело в том, что свойства объекта Gridlines применяются ко всем линиям сетки для определенной оси. Например, следующий код добавляет основные линии сетки оси категорий, а затем форматирует их:

```

ActiveChart.Axes(xlCategory, xlPrimary). _
    HasMajorGridlines = True
With ActiveChart.Axes(xlCategory, xlPrimary). _
    MajorGridlines.Border
        .Color = RGB(0, 0, 255)
        .LineStyle = xlDot
        .Weight = xlThick
    End With

```

Свойства, связанные с положением и размером осей

Объект Axis имеет набор свойств, связанных с его положением и размером. Все эти свойства предназначены только для чтения.

- свойство Height возвращает высоту оси (в точках);
- свойство Width возвращает ширину оси (в точках);

- свойство `Left` возвращает расстояние от левой границы оси до левой границы области диаграммы;
- свойство `Top` возвращает расстояние от верхней границы оси до верхней границы области диаграммы.

При этом ширина вертикальной оси, а также высота горизонтальной оси равны нулю.

Свойства, связанные с временными метками

Объект `Axis` имеет ряд свойств, связанных с временными метками. Свойство `MajorTickMark` возвращает или задает тип основных временных меток для указанной оси, а свойство `MinorTickMark` — для промежуточных временных меток. Оба этих свойства могут принимать значения констант из следующего перечисления:

```
Enum XlTickMark
    xlTickMarkNone = -4142    ' Метки отсутствуют
    xlTickMarkInside = 2     ' Метки находятся внутри диаграммы
    xlTickMarkOutside = 3    ' Метки находятся снаружи диаграммы
    xlTickMarkCross = 4      ' Метки пересекают ось
End Enum
```

Свойство `TickMarkSpacing` возвращает или определяет количество категорий или рядов данных между временными метками. Это свойство применяется только к осям категорий и рядам данных в трехмерных диаграммах. Для задания дополнительных меток на оси значений используются свойства `MajorUnit` и `MinorUnit`. Они будут рассмотрены чуть позднее.

Существует также несколько свойств, связанных с подписями к временным меткам. Свойство `TickLabels` возвращает объект `TickLabels`, который представляет собой подписи к временным меткам для указанной оси.

Свойство `TickLabelPosition` возвращает или определяет положение подписей к временным меткам на указанной оси. Оно может принимать значения констант из следующего перечисления:

```
Enum XlTickLabelPosition
    xlTickLabelPositionNone = -4142    ' Метки отсутствуют
    xlTickLabelPositionLow = -4134     ' Метки под диаграммой
    xlTickLabelPositionHigh = -4127    ' Метки над диаграммой
    xlTickLabelPositionNextToAxis = 4   ' Метки рядом с осью
End Enum
```

Свойство `TickLabelSpacing` возвращает или задает количество категорий или рядов данных, находящихся между подписями к временным меткам. Это свойство применяется только по отношению к осям категорий и рядов данных в трехмерных диаграммах. Подписи к временным меткам на оси значений приложение расставляет самостоятельно.

Объект `TickLabels`

Объект `TickLabels` представляет собой набор подписей к временным меткам для осей. Обратите внимание, что это единичный объект, а не семейство. В объектной модели Excel не существует объекта `TickLabel`. Соответственно, объект `TickLabels` оказывает влияние на все подписи к временным меткам для выбранной оси одновременно.

Перечислим наиболее существенные свойства объекта `TickLabels`:

- Свойство `Font` возвращает объект `Font`, определяющий шрифт подписи.
- Если свойство `AutoScaleFont` имеет заданное по умолчанию значение `True`, при изменении размера диаграммы происходит автоматическое масштабирование шрифта подписи.
- Свойство `NumberFormat` задает код числового формата для подписей. Если подписи имеют различный числовой формат, свойство возвращает значение `Null`. Так как в данном случае используются те же самые форматы кода, что и при форматировании ячеек листа, для их получения можно воспользоваться подпрограммой формирования, записи и сохранения клавиатурной макрокоманды.
- Свойство `Orientation` возвращает или задает ориентацию подписи к временным меткам и может принимать значения констант из следующего перечисления:

```
Enum xlTickLabelOrientation
    xlTickLabelOrientationUpward = -4171      ' Восходящая
    xlTickLabelOrientationDownward = -4170    ' Нисходящая
    xlTickLabelOrientationVertical = -4166     ' Вертикальная
    xlTickLabelOrientationHorizontal = -4128   ' Горизонтальная
    xlTickLabelOrientationAutomatic = -4105    ' Автоматическая
End Enum
```

Свойства, связанные с единицами измерения

Объект `Axis` имеет ряд свойств, связанных с единицами измерения и ценой деления шкалы.

- Свойства `MajorUnit` и `MinorUnit` возвращают или задают основные и промежуточные единицы определенной оси. Определение этого свойства автоматически приводит к тому, что соответствующему свойству `MajorUnitIsAuto` или `MinorUnitIsAuto` присваивается значение `False`.
- Если редактируемое логическое свойство `MajorUnitIsAuto` или `MinorUnitIsAuto` имеет значение `True`, происходит автоматическая установка основных или промежуточных единиц.
- Свойства `MajorUnitScale` и `MinorScaleUnit` возвращают или задают значение основной или промежуточной шкалы единиц для оси категорий в случае, если свойство `CategoryType` имеет значение `xlTimeScale`. Они могут принимать значения констант из следующего перечисления:

```
Enum xlTimeUnit
    xlDays = 0
    xlMonths = 1
    xlYears = 2
End Enum
```

- Свойства `MaximumScale` и `MinimumScale` возвращают или задают максимальное или минимальное значение оси в формате `Double`.
- Если логическое редактируемое свойство `MaximumScaleIsAuto` или `MinimumScaleIsAuto` имеет значение `True`, происходит вычисление максимального или минимального значения для определенной оси.

- Свойство `ScaleType` возвращает или определяет тип масштабирования значений вдоль оси. Это свойство применимо только к оси значений и может принимать значения констант из следующего перечисления:

```
Enum XlScaleType
    xlScaleLogarithmic = -4133 ' Десятичный логарифм
    xlScaleLinear = -4132 ' Линейный закон
End Enum
```

Для иллюстрации рассмотрим следующий код:

```
With ActiveChart.Axes(xlValue, xlPrimary)
    .MajorUnit = 1
    .MaximumScale = 5
End With.
```

в результате выполнения которого ось, изображенная на рис. 21.12, отображается слева. С другой стороны, код:

```
With ActiveChart.Axes(xlValue, xlPrimary)
    .MajorUnit = 5
    .MaximumScale = 3
End With
```

приводит к тому, что ось на рис. 21.12 отображается справа.

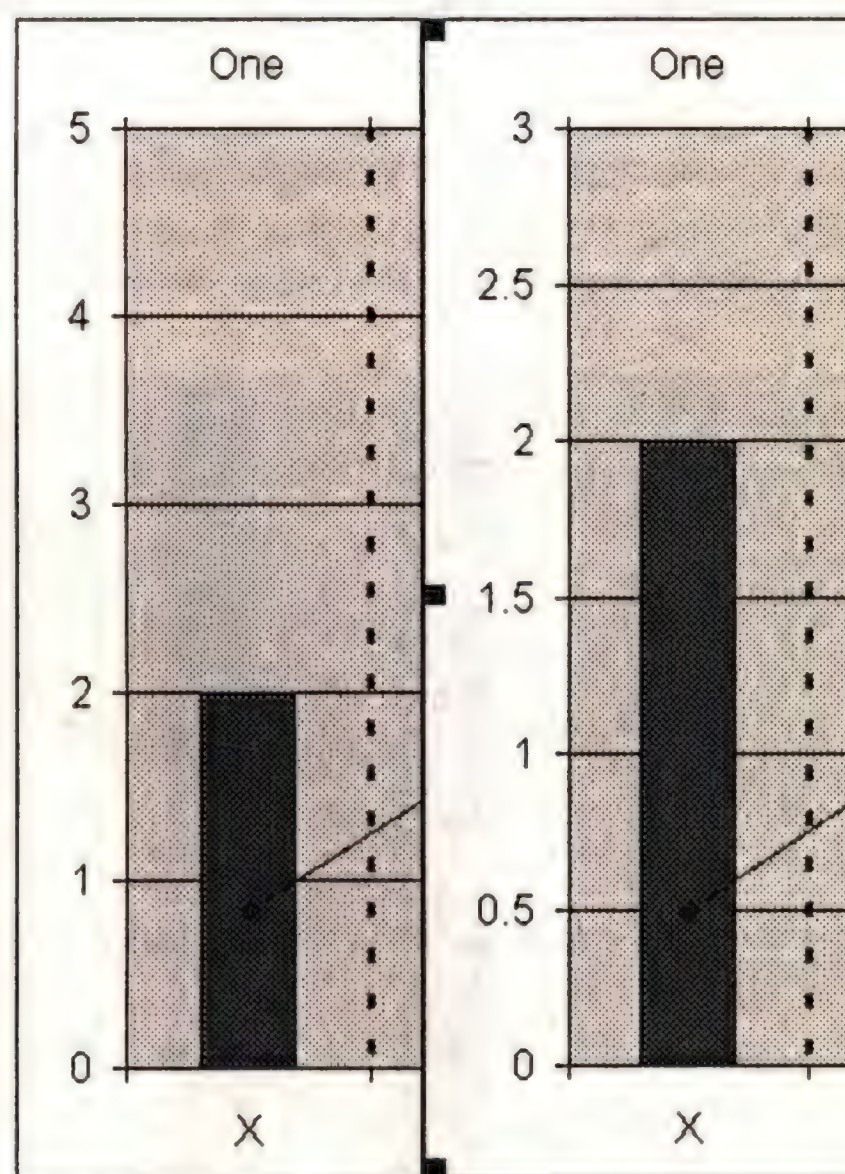


Рис. 21.12. Единицы и масштаб осей

Свойство `ReversePlotOrder`

В результате присваивания логическому редактируемому свойству `ReversePlotOrder` значения `True` изменяются направления осей. Например, для диаграммы, изображенной на рис. 21.13, код:

```
ActiveChart.Axes(xlValue, xlPrimary).ReversePlotOrder = True
```


приводит к созданию диаграммы, изображенной на рис. 21.14, у которой нумерация первичной оси значений начинается в направлении сверху вниз.

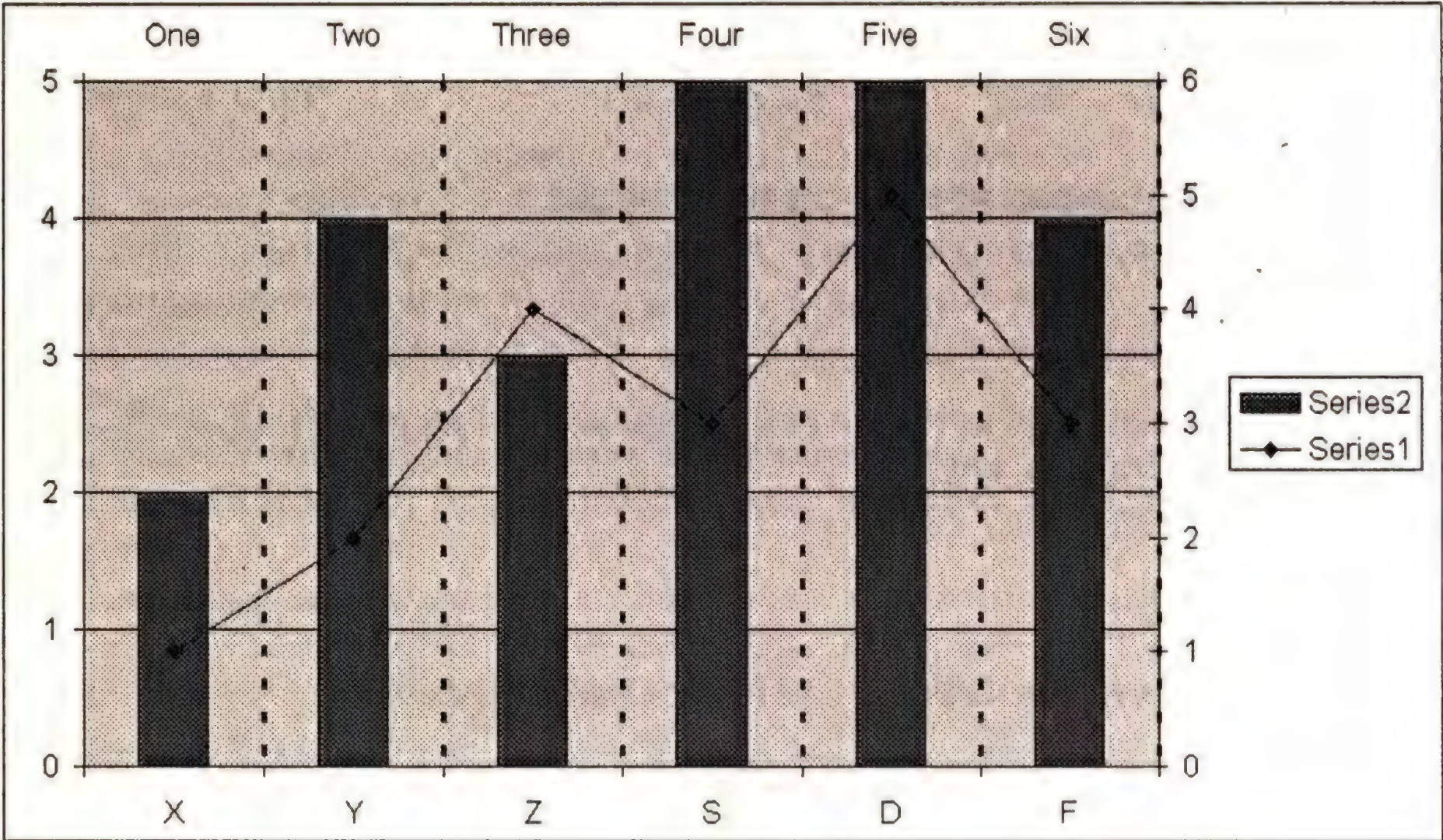


Рис. 21.13. Диаграмма перед применением свойства ReversePlotOrder

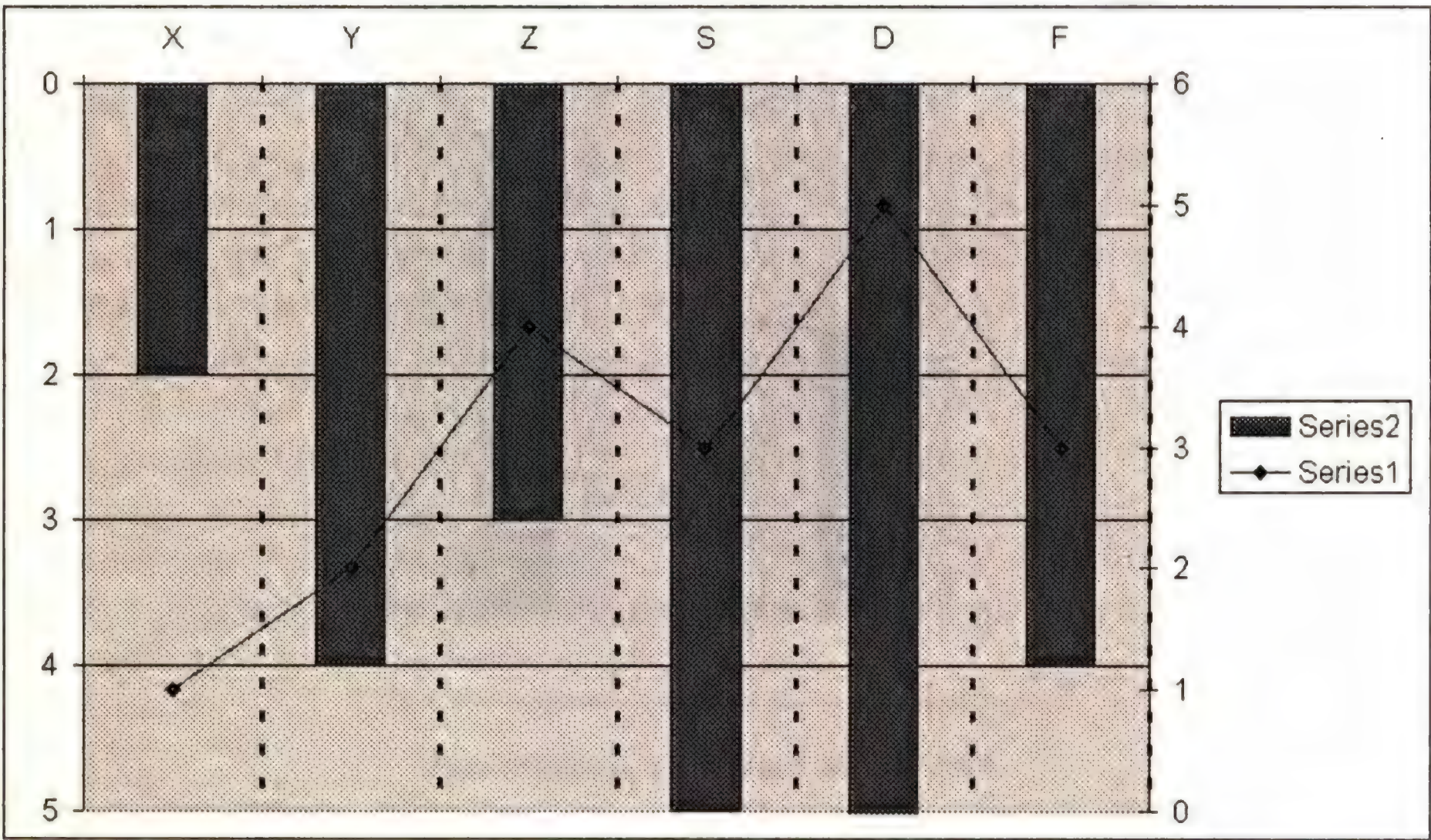


Рис. 21.14. Диаграмма после применения свойства ReversePlotOrder

Свойство Type

Свойство Type возвращает или определяет тип оси. Оно может принимать значения констант из следующего перечисления:


```
Enum XlAxisType
    xlCategory = 1      ' Ось категорий
    xlValue = 2         ' Ось значений
    xlSeriesAxis = 3    ' Ось рядов
End Enum
```

Объект ChartArea

Объект ChartArea представляет собой область фона диаграммы. Если обратиться к рис. 21.1, область двумерной диаграммы будет включать в числе прочего также оси, заголовок диаграммы, заголовки осей и легенды. В то же время область трехмерной диаграммы содержит ее заголовок и легенду, но не включает область, реально занятую диаграммой.

Объект ChartArea включает четырех потомков — Border, ChartFillFormat, Font и Interior. С их помощью можно задавать характеристики всей диаграммы. Все эти объекты обсуждались ранее в связи с заголовками осей.

К сожалению, в справочной документации недостаточно ясно сформулирована разница между объектом ChartFillFormat (определяет заливку элементов диаграммы) и объектом Interior. Соответственно, пользователю предстоит выяснить ее самостоятельно экспериментальным путем. Несмотря на то что оба этих объекта имеют одно и то же свойство Pattern, их остальные свойства и методы отличаются друг от друга. Создается впечатление, что объект ChartFillFormat связан с градиентной и текстурной заливкой, в то время как объект Interior — со сплошной и трафаретной заливкой.

Для иллюстрации рассмотрим строку кода, приводящую к заливке области диаграммы красным цветом (параметр ColorIndex имеет значение 3):

```
ActiveChart.ChartArea.Interior.ColorIndex = 3
```

Следующий код создает градиентную заливку области диаграммы с переходом от красного цвета наверху к фиолетовому цвету снизу:

```
With ActiveChart.ChartArea.Fill
    .Visible = True
    .ForeColor.SchemeColor = 3
    .BackColor.SchemeColor = 7
    .TwoColorGradient Style:=msoGradientHorizontal, Variant:=1
End With
```

Объект ChartGroup

Как вы уже знаете, диаграмма Excel может содержать несколько рядов данных. Ранее уже отмечалось, что каждый ряд (то есть объект Series) имеет свойство ChartType, которое используется для определения типа диаграммы.

Объект ChartGroup представляет собой набор рядов данных, изображенных на одной диаграмме одним и тем же способом. Имейте в виду, что объект ChartGroup не является семейством. Для доступа к индивидуальным объектам Series, входящим в объект ChartGroup, используется свойство SeriesCollection, о котором мы поговорим чуть позднее.

Объекты `ChartGroup`, принадлежащие к одной и той же диаграмме, относятся к семейству `ChartGroups` объекта `Chart`. Доступ к этому семейству осуществляется с помощью свойства `ChartGroups`.

Объектная модель Excel дает возможность получить «подсемейства» семейства `ChartGroups`, соответствующие основным типам диаграмм (график, гистограмма, линейчатая диаграмма и т. п.). Например, метод `ColumnGroups`, примененный к двумерной диаграмме, возвращает семейство всех объектов `ChartGroup`, соответствующих различным форматам гистограммы. Обратите внимание на синтаксис метода: `ChartObject.ColumnGroups`

Можно также получить доступ к одному объекту `ChartGroup` из этого семейства с помощью строки:

`ChartObject.ColumnGroups(Index)`

Имейте в виду, что семейства `ColumnChartGroups` не существует в природе. Метод `ColumnGroups` возвращает семейство `ChartGroups`. Если же требуется вернуть полное семейство, используется код:

`ChartObject.ChartGroups`

Для иллюстрации рассмотрим диаграмму, изображенную на рис. 21.15, содержащую два ряда данных. Несмотря на то что каждый ряд принадлежит к типу `Line` (График), их подтипы различаются. Один ряд имеет маркеры данных, а второй — нет. Соответственно, код:

`ActiveChart.LineGroups.Count`

возвращает значение 2, так как в данном случае существуют два разных объекта `ChartGroup`, входящие в семейство `LineGroups`.

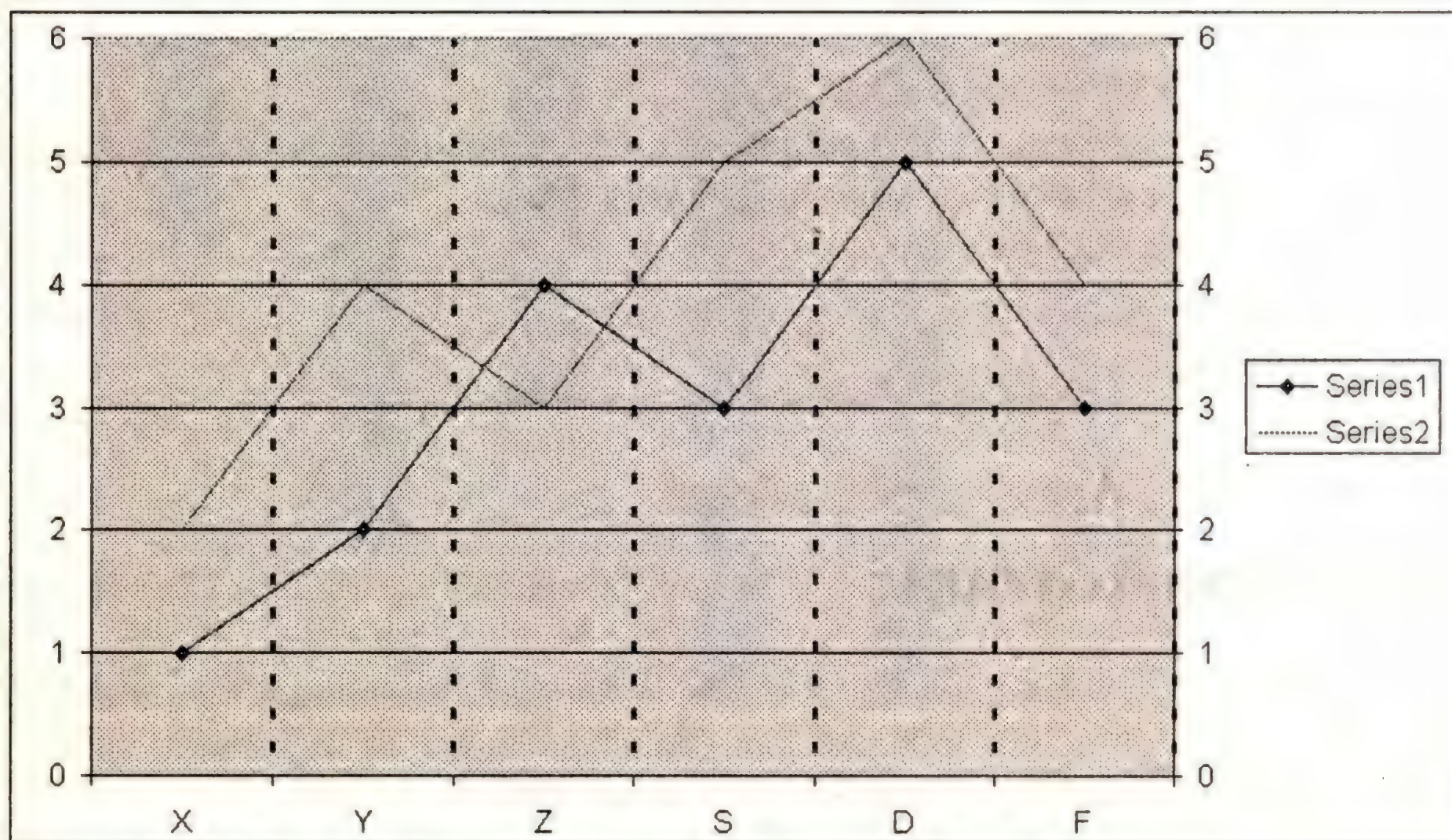


Рис. 21.15. Группы диаграмм

Объект `Chart` включает следующие методы, возвращающие соответствующие наборы объектов `ChartGroup`:

- AreaGroups
- BarGroups
- ColumnGroups
- DoughnutGroups
- LineGroups
- PieGroups
- RadarGroups
- XYGroups

Существуют также свойства и методы объекта Chart, возвращающие группы в случае трехмерных диаграмм: Area3DGroup, Bar3DGroup, Column3DGroup, Line3DGroup, Pie3DGroup и SurfaceGroup. Все они возвращают объект ChartGroup.

Часть объектной модели Excel, связанная с объектами ChartGroup, показана на рис. 21.16.

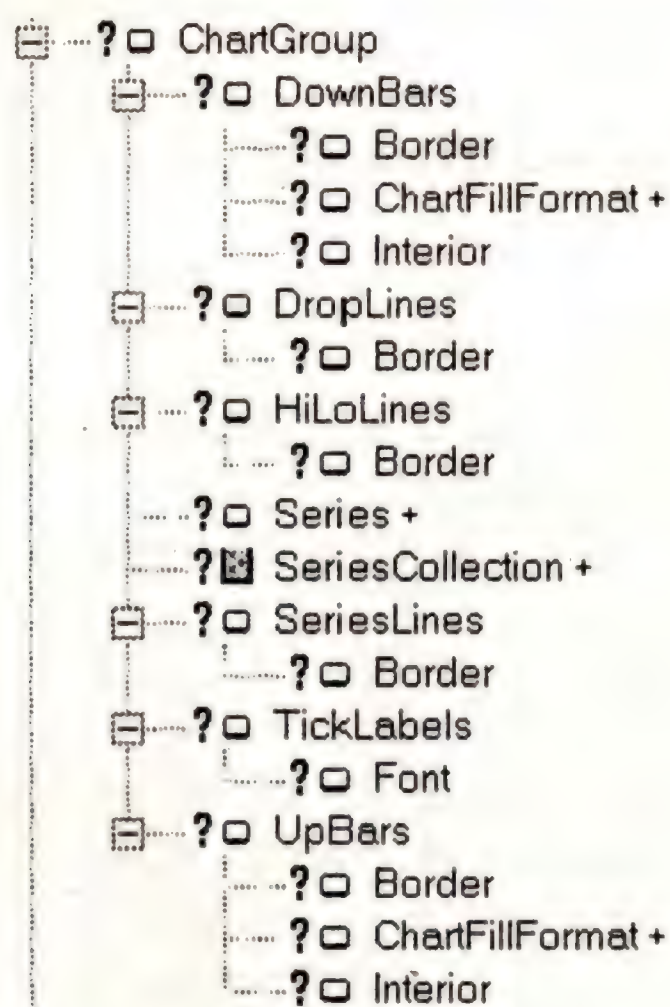


Рис. 21.16. Объект ChartGroup

Объекты UpBars и DownBars

Объекты UpBars и DownBars, которые встречаются исключительно на двухмерных диаграммах, показаны на рис. 21.17. При использовании этих объектов Excel рисует в каждой точке данных столбик, ведущий от первого к последнему ряду. Если значение первого ряда в данной точке больше последнего, на диаграмме отображается объект DownBars (черный). В противном случае отображается объект UpBars (белый).

Вот код, генерирующий эти объекты:

```

With ActiveChart.LineGroups(1)
    .HasUpDownBars = True
    .UpBars.Interior.Color = RGB(255, 255, 255)
    .DownBars.Interior.Color = RGB(0, 0, 0)
End With
  
```

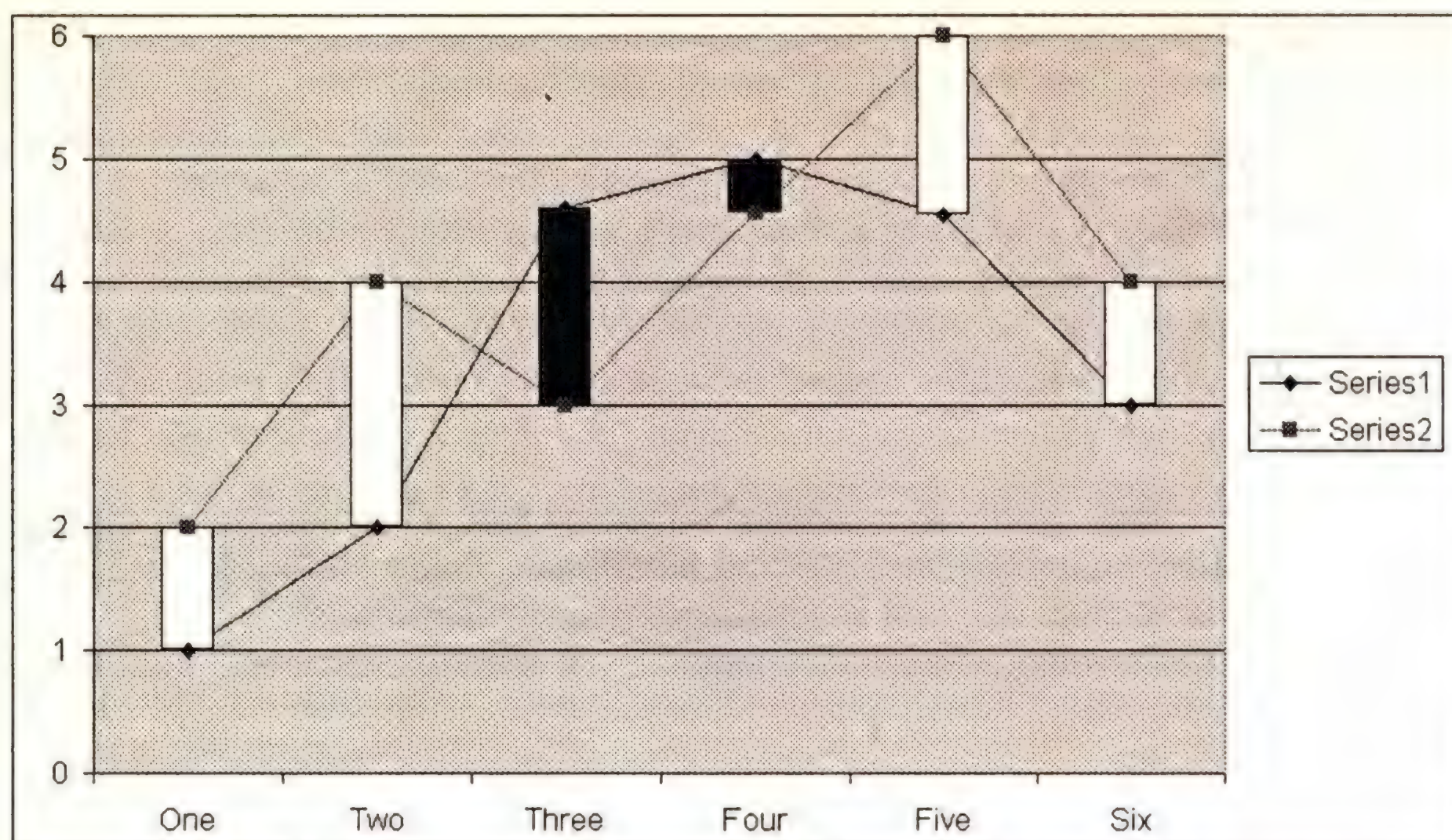



Рис. 21.17. Объект UpBars окрашен в белый цвет, а объект DownBars — в черный

Объект DropLines

Объект DropLines представляет собой вертикальные линии, опущенные из нанесенной точки до оси категорий. Чтобы сделать их видимыми, требуется присвоить свойству HasDropLines объекта ChartGroup значение True.

Объект HiLoLines

Объект HiLoLines представлен на рис. 21.18. Он представляет собой вертикальные линии между максимальным и минимальным значением точки данных. Чтобы сделать их видимыми, требуется присвоить свойству HasHiLoLines объекта ChartGroup значение True. Объект HiLoLines используется только на двухмерных диаграммах.

Семейство SeriesCollection и объект Series

Свойство SeriesCollection объекта ChartGroup возвращает семейство SeriesCollection всех объектов Series, принадлежащих к данной группе. Объект Series будет подробнее рассматриваться чуть позже.

Объект SeriesLines

Объект SeriesLines показан на рис. 21.19. Он используется в обычных и столбцовых линейчатых диаграммах с накоплением и в аналогичных гистограммах. Объекты SeriesLines объединяют слои в стопках, относящихся к одному и тому же ряду данных. Чтобы сделать их видимыми, требуется присвоить свойству HasSeriesLines значение True.

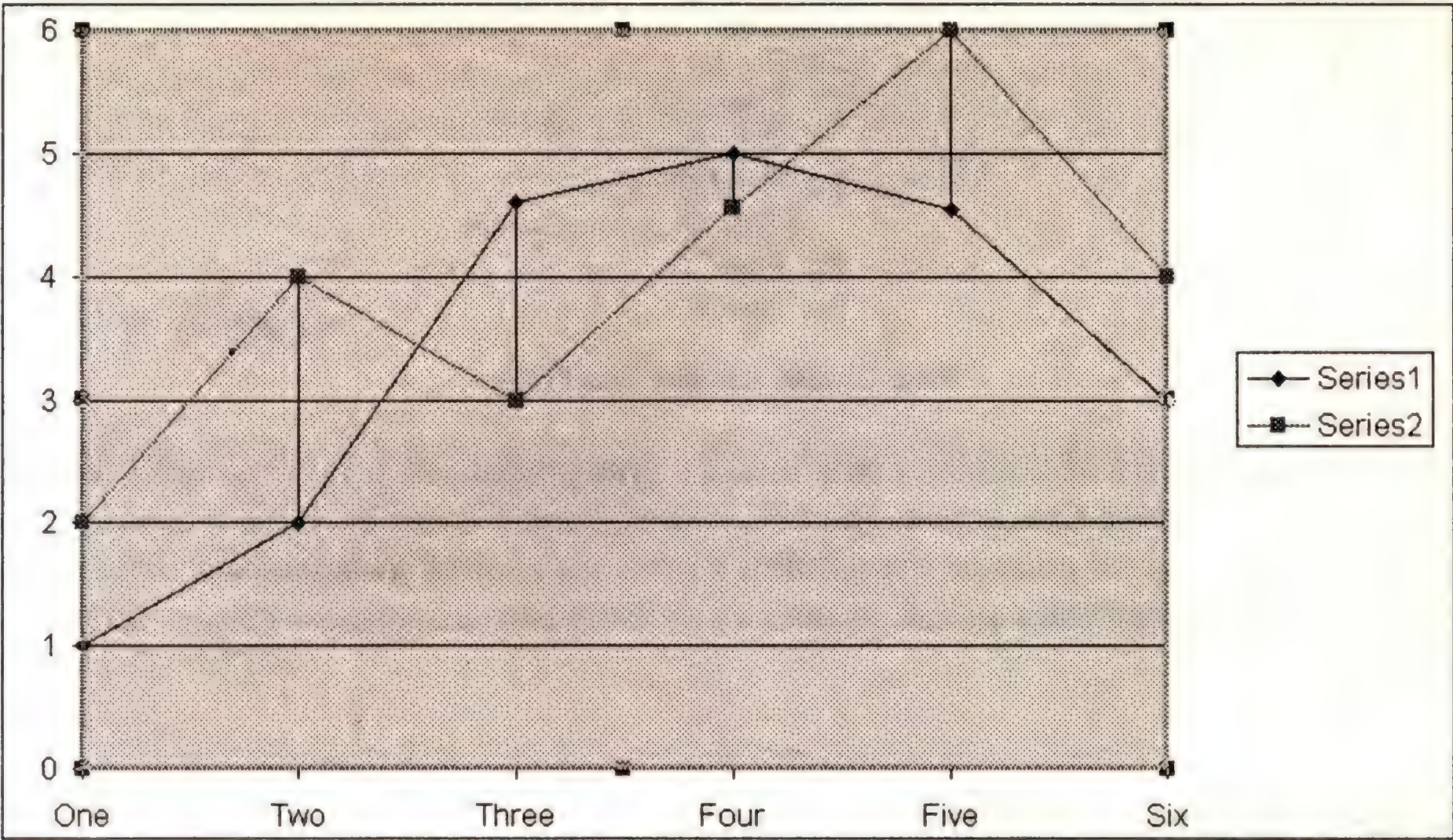


Рис. 21.18. Объект HiLoLines

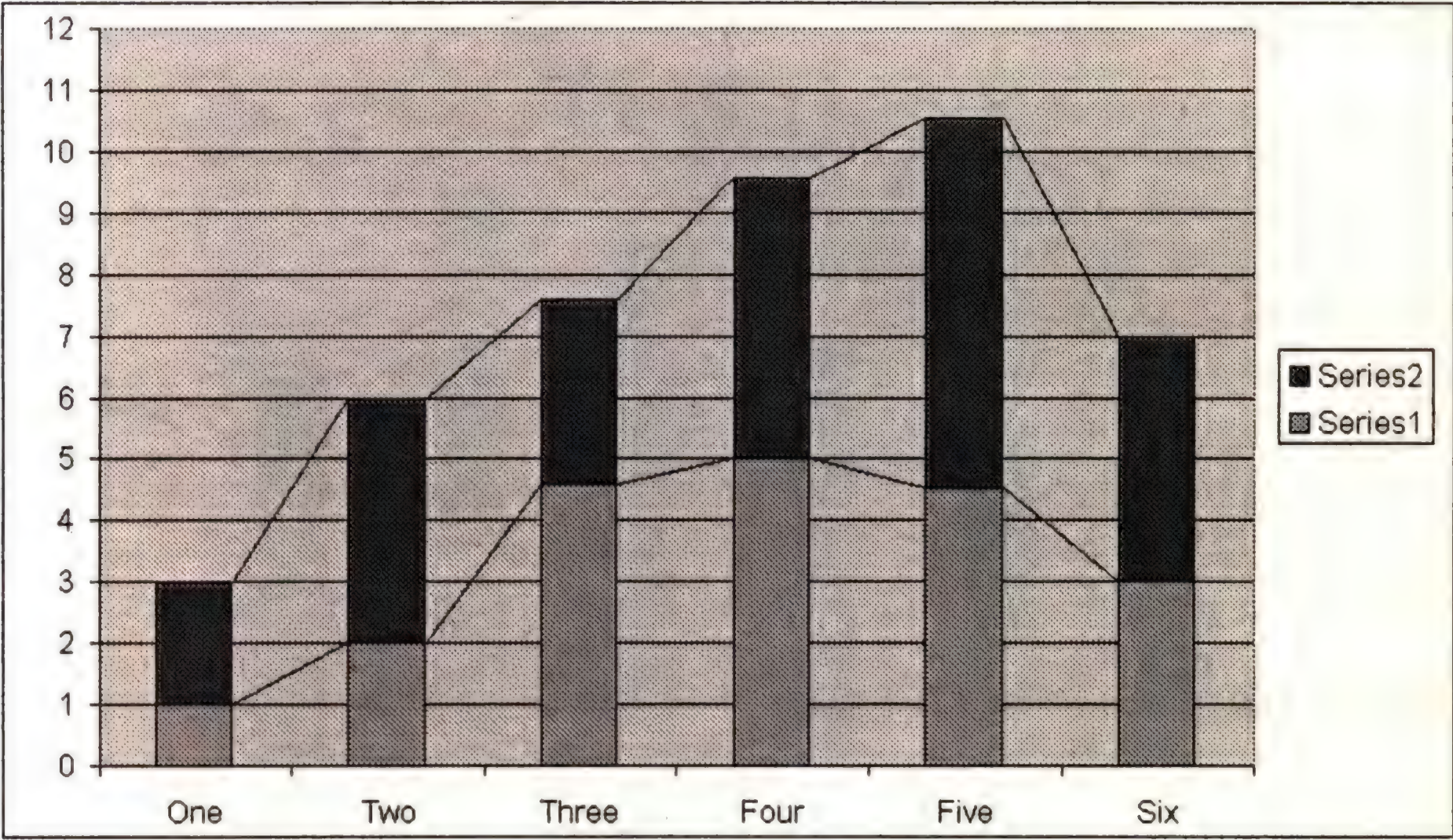


Рис. 21.19. Объект SeriesLines

Объект ChartTitle

Объект ChartTitle представляет собой заголовок диаграммы. Соответствующая ему часть объектной модели Excel показана на рис. 21.20.

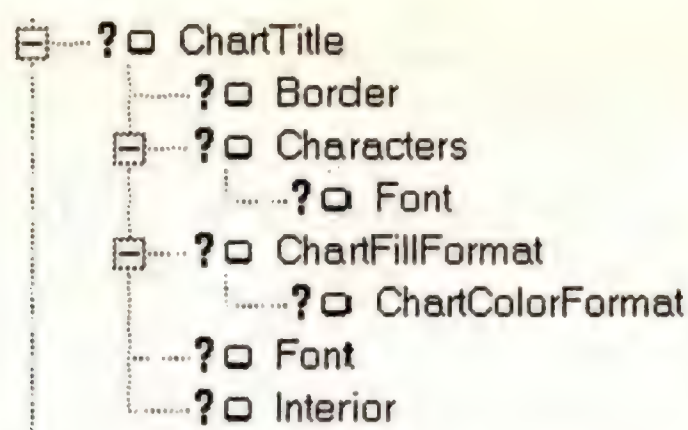


Рис. 21.20. Объект ChartTitle

Как и ряд других объектов, связанных с диаграммами, объект ChartTitle имеет ряд потомков: Border, Characters, ChartFillFormat, Font и Interior. Они используются для форматирования соответствующей части заголовка диаграммы. Эти объекты уже рассматривались ранее, поэтому сейчас мы останавливаться на них не будем.

Объект DataTable

На рис. 21.21 вместе с диаграммой показана таблица данных. Эта таблица представлена объектом DataTable.

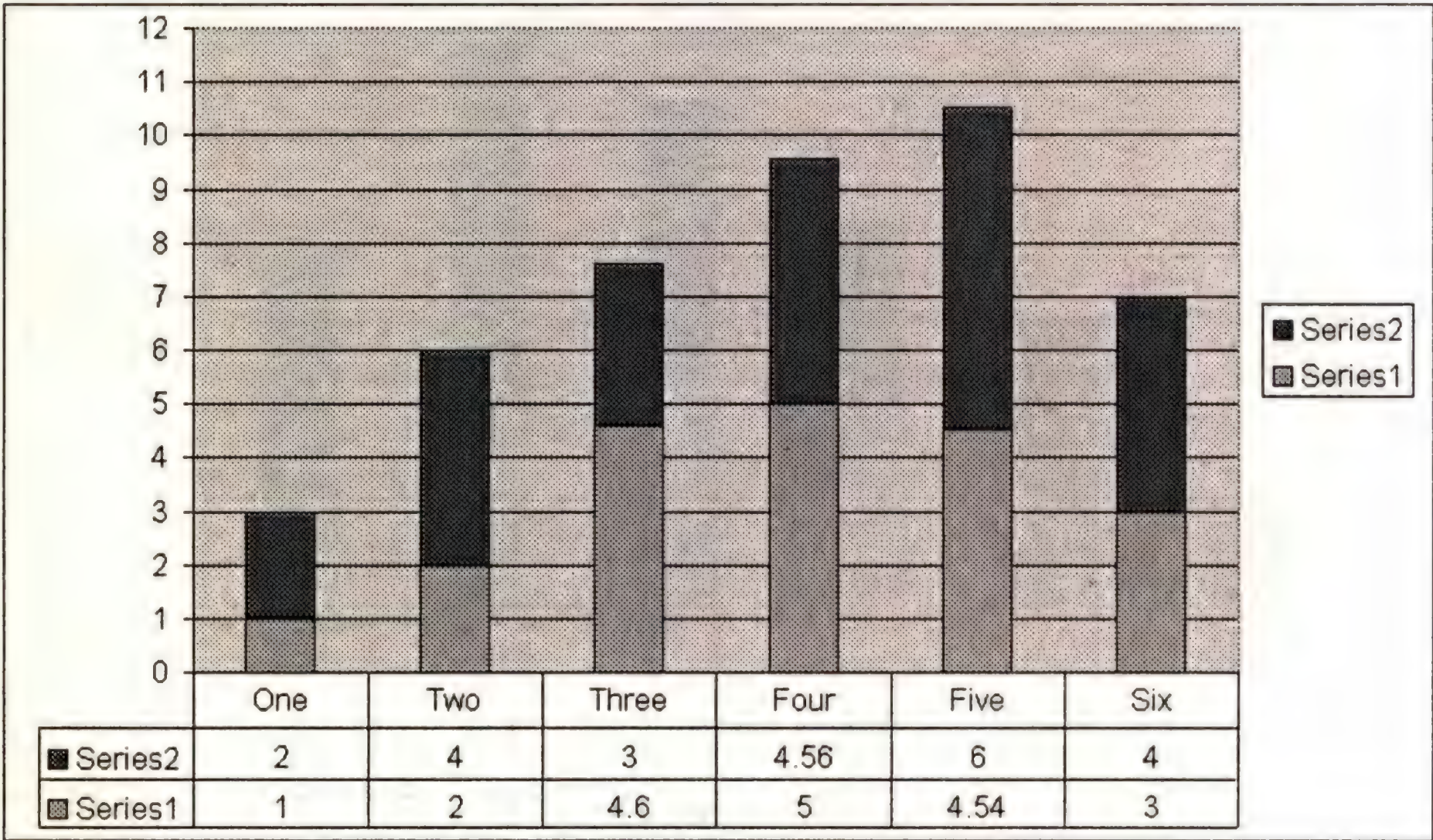


Рис. 21.21. Таблица данных

Объект Chart имеет свойство HasDataTable. Чтобы сделать видимой таблицу данных, достаточно присвоить этому свойству значение True. В действительности таблица данных, показанная на рис. 21.21, и ее граница созданы с помощью кода:

```
ActiveChart.HasDataTable = True
ActiveChart.DataTable.HasBorderOutline = True
```


Объект DataTable имеет ряд свойств, назначение которых понятно из их названий: AutoScaleFont, Border, Font, HasBorderHorizontal, HasBorderOutline, HasBorderVertical и ShowLegendKey. Последнее свойство определяет отображение небольших квадратов в левой части таблицы, изображенной на рис. 21.21.

Объект Floor

Объект Floor относится исключительно к трехмерным диаграммам и представляет их основание. Например, следующий код:

```
ActiveChart.Floor.Interior.Pattern = xlPatternChecker
```

заливает основание трехмерной диаграммы рисунком шахматного поля, как показано на рис. 21.22.

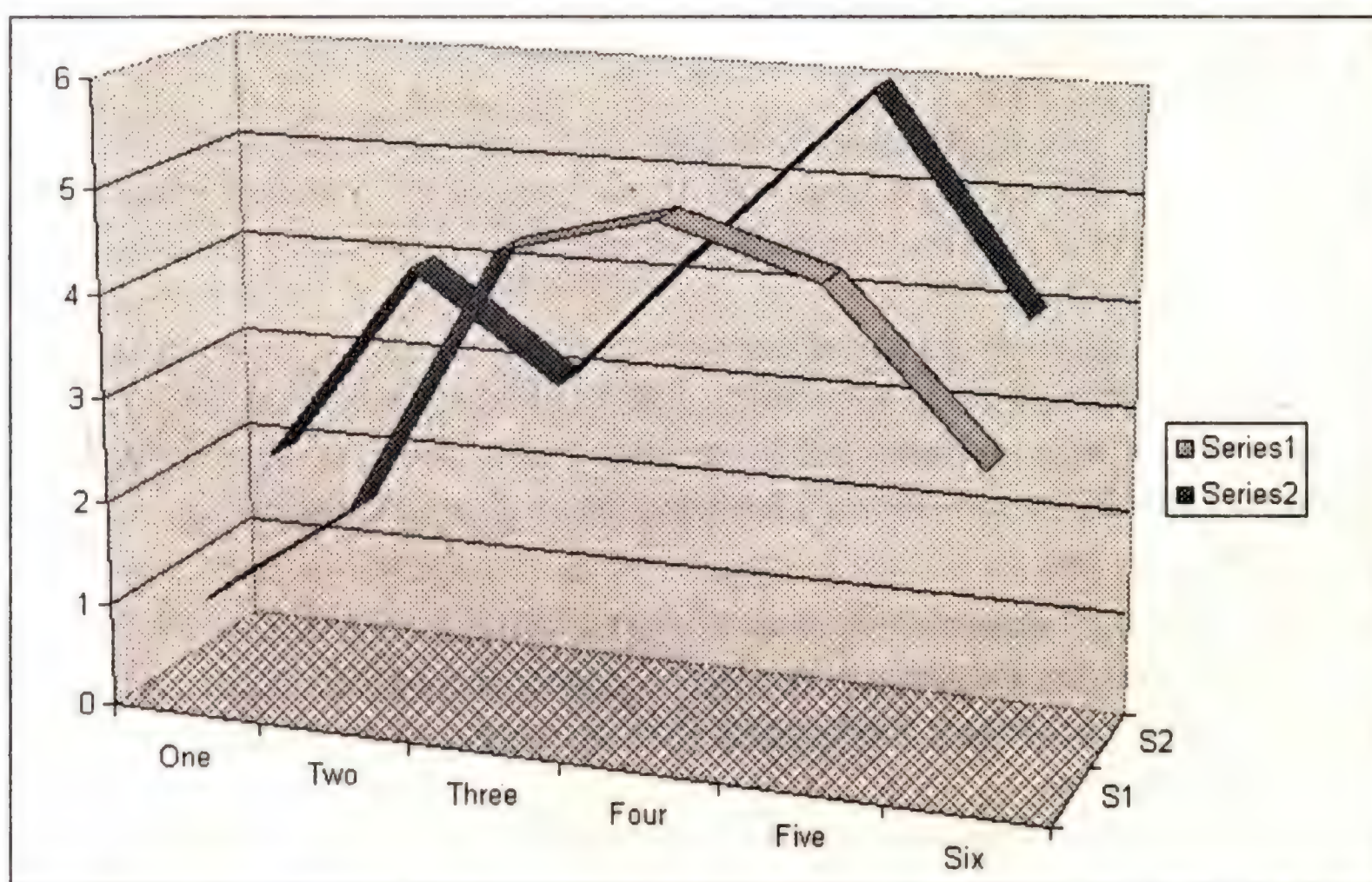


Рис. 21.22. Основание трехмерной диаграммы с рисунком шахматной доски

Объект Floor имеет трех потомков: Border, ChartFillFormat и Interior.

Объект Legend

Объект Legend представляет собой легенду, то есть подпись, определяющую узор, цвета рядов или категорий данных на диаграмме. Этот объект и его потомки показаны на рис. 21.23.

Объект Chart имеет свойство Legend, возвращающее объект Legend для рассматриваемой диаграммы. Разумеется, потомками объекта Legend являются объекты Border, ChartFillFormat, Font и Interior, выполняющие те же самые функции, что и в случае с объектами ChartTitle, AxisTitle, ChartArea и т. п.

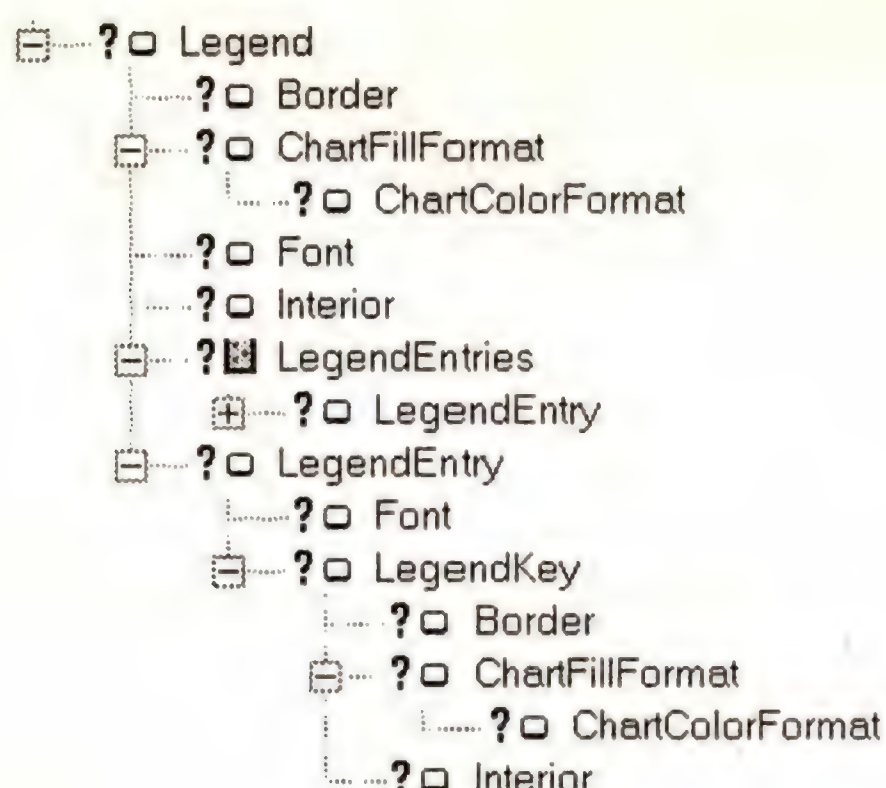


Рис. 21.23. Объект Legend

Объект LegendEntry

Легенды диаграммы, изображенной на рис. 21.1, состоят из двух элементов. Эти элементы являются объектами `LegendEntry` и хранятся в семействе `LegendEntries` объекта `Legend`. Доступ к этому семейству осуществляется с помощью свойства `LegendEntries` объекта `Chart`.

Каждая легенда состоит из двух частей: текст и маркер. При этом невозможно изменить название ряда данных в легенде средствами объекта `LegendEntry`. Для этого нужно применять свойство `Name` объекта `Series`. Впрочем, объект `LegendEntry` обладает свойством `Font`, которое позволяет изменять шрифт надписи:

```
ActiveChart.Legend.Font.Italic = True
```

Кроме того, объект `LegendEntry` может быть удален. Но после этого единственным способом его восстановления является удаление всей легенды и ее повторное создание. Для этого свойству `HasLegend` нужно сначала присвоить значение `False`, а потом — значение `True`.

Легенды невозможно форматировать по частям, а также изменять их положение и размер.

Напоследок заметим, что не существует способа непосредственно вернуть ряд данных, соответствующий легенде.

Объект LegendKey

Маркер легенды представлен объектом `LegendKey`. Его потомками являются объекты `Border`, `ChartFillFormat` и `Interior`.

Важно помнить, что форматирование объекта `LegendKey` приводит к автоматическому форматированию ряда данных, представленных указанной легендой. Другими словами, ряд данных всегда совпадает с маркером легенды.

Например, следующий код форматирует первый ряд данных и соответствующий ему маркер легенды, заливая фон красным и утолщая границу:

```
With ActiveChart.Legend.LegendEntries(1).LegendKey
    .Interior.ColorIndex = 3
    .Border.Weight = xlThick
End With
```


Объект PageSetup

Объект PageSetup представляет все атрибуты страницы, связанные с выводом диаграммы на печать. Свойства и методы этого объекта перечислены в табл. 21.3.

Таблица 21.3. Свойства и методы объекта PageSetup

Application	FitToPagesWide	PrintErrors<v10>
BlackAndWhite	FooterMargin	PrintGridlines
BottomMargin	HeaderMargin	PrintHeadings
CenterFooter	LeftFooter	PrintNotes
CenterFooterPicture<v10>	LeftFooterPicture<v10>	PrintQuality
CenterHeader	LeftHeader	PrintTitleColumns
CenterHeaderPicture<v10>	LeftHeaderPicture<v10>	PrintTitleRows
CenterHorizontally	LeftMargin	RightFooter
CenterVertically	Order	RightFooterPicture<v10>
ChartSize	Orientation	RightHeader
Creator	PaperSize	RightHeaderPicture<v10>
Draft	Parent	RightMargin
FirstPageNumber	PrintArea	TopMargin
FitToPagesTall	PrintComments	Zoom

Вот пример кода, который определяет поля активной диаграммы, а затем осуществляет ее предварительный просмотр:

```
With ActiveChart.PageSetup
    .LeftMargin = Application.InchesToPoints(0.5)
    .RightMargin = Application.InchesToPoints(0.75)
    .TopMargin = Application.InchesToPoints(1.5)
    .BottomMargin = Application.InchesToPoints(1)
    .HeaderMargin = Application.InchesToPoints(0.5)
    .FooterMargin = Application.InchesToPoints(0.5)
End With
ActiveChart.PrintPreview
```

Объект PlotArea

Объект PlotArea представляет собой область, реально занятую диаграммой. В случае двухмерных диаграмм сюда входят маркеры данных, линии сетки, подписи к данным и прочие элементы диаграммы, кроме ее осей. Для трехмерных диаграмм в объект PlotArea включаются стены и основание диаграммы, ее оси, заголовки осей и метки осей.

Объект PlotArea имеет потомков Border, ChartFillFormat и Interior, которые используются для форматирования соответствующих элементов. Кроме того, он имеет также свойства Top, Left, Height и Width, задающие размер и положение объекта PlotArea внутри объекта ChartArea. Существует ряд ограничений на значения этих свойств. Например, невозможно присвоить свойству Top такое значение, чтобы нижняя часть объекта PlotArea оказалась под нижней частью объекта ChartArea.

Объект Series

Объект Series представляет собой ряд данных, нанесенных на диаграмму. Соответствующая ему часть объектной модели Excel показана на рис. 21.24.

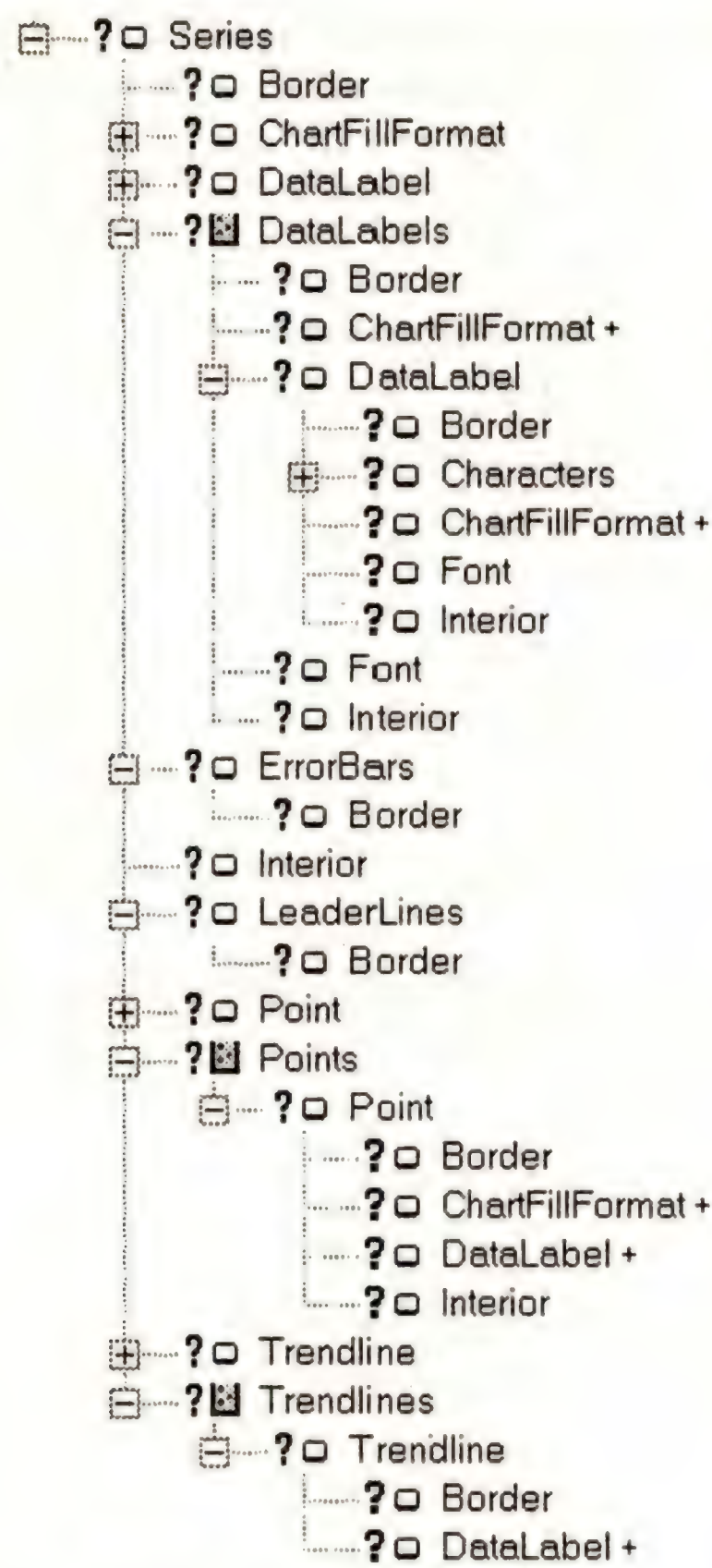


Рис. 21.24. Объект Series и его потомки

В число потомков объекта Series входят объекты Border, ChartFillFormat и Interior, назначение которых уже упоминалось ранее. Рассмотрим ряд других дочерних объектов.

Каждый объект Series, относящийся к конкретной диаграмме, является частью семейства SeriesCollection. Это семейство возвращается методом SeriesCollection объекта Chart.

Свойства и методы объекта Series перечислены в табл. 21.4.

Таблица 21.4. Свойства и методы объекта Series

Application	ErrorBars	MarkerForegroundColorIndex
ApplyCustomType	Explosion	MarkerSize
ApplyDataLabels	Fill	MarkerStyle

ApplyPictToEnd	Formula	Name
ApplyPictToFront	FormulaLocal	Parent
ApplyPictToSides	FormulaR1C1	Paste
AxisGroup	FormulaR1C1Local	PictureType
BarShape	Has3DEffect	PictureUnit
Border	HasDataLabels	PlotOrder
BubbleSizes	HasErrorBars	Points
ChartType	HasLeaderLines	Select
ClearFormats	Interior	Shadow
Copy	InvertIfNegative	Smooth
Creator	LeaderLines	Trendlines
DataLabels	MarkerBackgroundColor	Type
Delete	MarkerBackgroundColorIndex	Values
ErrorBar	MarkerForegroundColor	Xvalues

Добавление новых рядов данных

Чтобы добавить на диаграмму новый ряд данных, используется метод Add объекта SeriesCollection. Вот его синтаксис:

```
SeriesCollectionObject.Add(Source, Rowcol, _
    SeriesLabels, CategoryLabels, Replace)
```

Параметр *Source* задает новые данные в виде объекта Range.

Необязательный параметр *Rowcol* указывает, как именно расположены данные: в виде строк или в виде столбцов. Он может принимать значения констант из следующего перечисления:

```
Enum XlRowCol
    xlRows = 1
    xlColumns = 2 ' По умолчанию
End Enum
```

Необязательный параметр *SeriesLabels* применяется только в том случае, когда параметр *Source* представляет собой диапазон, а не массив. Если первая строка или столбец содержит имя ряда данных, этот параметр имеет значение True. Если же в первой строке или в столбце содержится первая точка ряда данных, этот параметр имеет значение False. При пропуске аргумента *SeriesLabels* Excel пытается определить, где находится имя ряда данных, по содержимому первой строки или столбца. (Как уже не раз упоминалось, оставлять подобные вещи на усмотрение приложения не рекомендуется.)

Необязательный параметр *CategoryLabels* также применяется только в тех случаях, когда параметр *Source* является диапазоном. Он имеет значение True, если первая строка или столбец содержат название метки категории. Пропуск этого аргумента приводит к тому, что приложение пытается определить положение метки категории, исходя из содержимого первой строки или столбца.

Смысл необязательного параметра *Replace* можно объяснить следующим образом: если параметр *CategoryLabels* имеет значение True и параметр *Replace* имеет значение True, указанная пользователем категория заменяет категорию, существующую для выделенной серии в данный момент. По умолчанию параметр *Replace* имеет значение False, и замены категории не происходит.

Для иллюстрации рассмотрим лист, показанный на рис. 21.25.

	A	B	C
1		Average	Std. Dev.
2	A	1.23	2
3	B	2	4
4	C	4.6	3
5	D	4.43	4.56
6	E	4.54	6
7	F	3	4

Рис. 21.25. Лист, предназначенный для иллюстрации применения метода Add

Следующий код приведет к появлению диаграммы, изображенной на рис. 21.26:

```
Dim co As ChartObject
Set co = ActiveSheet.ChartObjects. _
    Add(100, 100, 300, 200)
co.Chart.ChartType = xlColumnClustered
co.Chart.SeriesCollection.Add
    Source:=ActiveSheet.Range("A1:C7"), _
    Rowcol:=xlColumns, SeriesLabels:=True, _
    CategoryLabels:=True
```

Обратите внимание на то, что метки рядов находятся в первой строке, а метки категорий — в первом столбце.

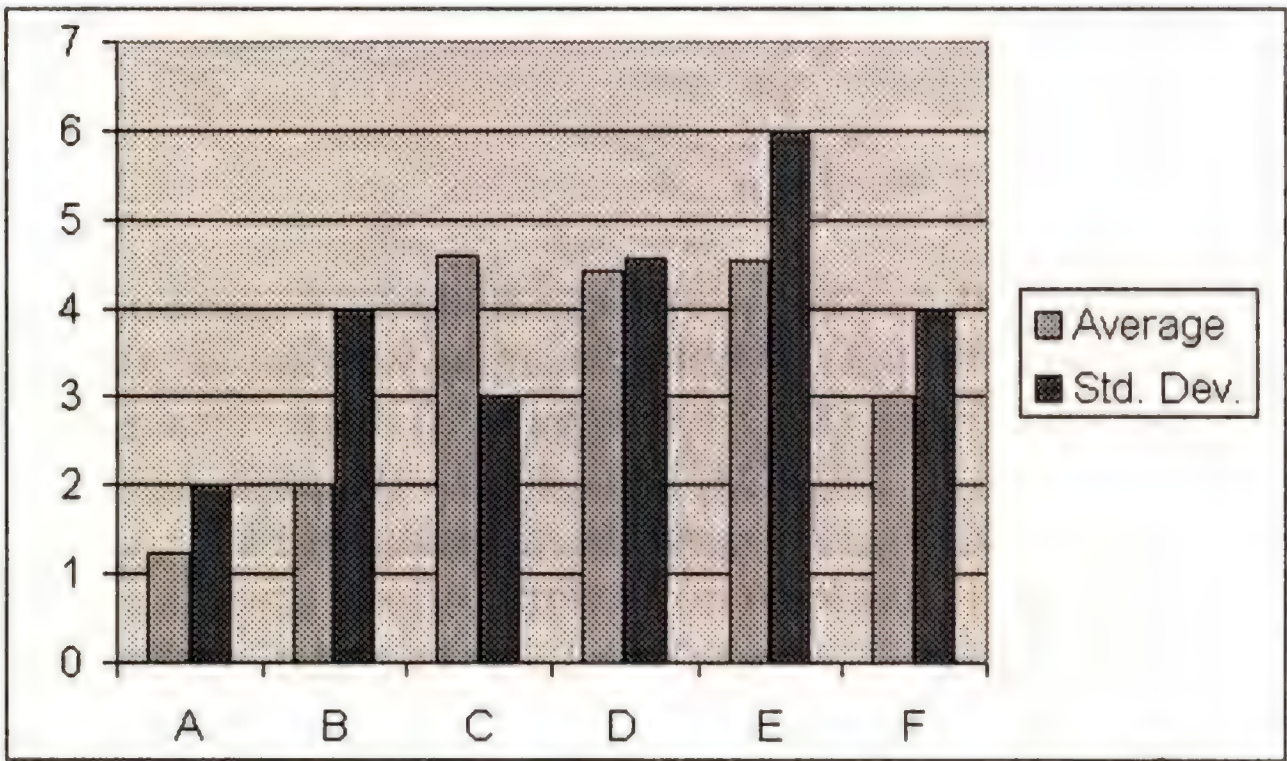


Рис. 21.26. Диаграмма, полученная с помощью метода Add

Объект DataLabel

Объект DataLabel представляет собой поле рядом с точкой на диаграмме, в котором отображается значение точки, имя ряда данных или текстовая строка. Каждый из объектов Series включает семейство DataLabels, которое содержит по одному объекту DataLabel для каждой точки в ряду. Семейство DataLabels возвращается методом DataLabels:


```
If ActiveChart.SeriesCollection(1). _
    HasDataLabels Then
    MsgBox ActiveChart.SeriesCollection(1). _
        DataLabels.Count
End If
```

Если для рассматриваемого ряда данных объект DataLabel отсутствует, метод DataLabels приведет к появлению сообщения об ошибке. Поэтому перед вызовом данного метода требуется проверять наличие объекта DataLabel. Это делается с помощью свойства HasDataLabels объекта Series.

Отображением объекта DataLabel управляет свойство HasDataLabel соответствующего объекта Point (о нем мы поговорим чуть позже). Соответственно, код:

```
ActiveChart.SeriesCollection(1).Points(1).HasDataLabel = False
```

запрещает отображение объекта DataLabel для первой точки данных в ряду.

Чтобы сделать видимым или скрыть объект DataLabel, можно воспользоваться методом ApplyDataLabels. Кроме того, он изменяет тип меток. Данный метод имеет следующий синтаксис:

```
expression.ApplyDataLabels(Type, LegendKey),
```

где выражение *expression* может возвращать объекты Chart, Point или Series. Если метод применяется к объекту Chart, он влияет на метки данных для всех представленных на диаграмме рядов одновременно.

Параметр *Type* определяет тип метки данных и может принимать значения констант из следующего перечисления:

```
Enum xlDataLabelsType
    xlDataLabelsShowNone = -4142
    xlDataLabelsShowValue = 2
    xlDataLabelsShowPercent = 3
    xlDataLabelsShowLabel = 4
    xlDataLabelsShowLabelAndPercent = 5
    xlDataLabelsShowBubbleSizes = 6
End Enum
```

Если необязательному параметру *LegendKey* присвоить значение True, рядом с каждой точкой данных будет фигурировать ключ легенды. По умолчанию этот параметр имеет значение False. Пример легенд точек данных показан на рис. 21.27.

Свойства и методы объекта DataLabel перечислены в табл. 21.5.

Таблица 21.5. Свойства и методы объекта DataLabel

Application	Font	ReadingOrder
AutoScaleFont	HorizontalAlignment	Select
AutoText	Interior	Shadow
Border	Left	ShowLegendKey
Caption	Name	Text
Characters	NumberFormatLinked	Top
Creator	Orientation	Type
Delete	Parent	VerticalAlignment
Fill	Position	

На линиях тренда (о них мы поговорим чуть позже) свойство DataLabel возвращает метку, специфичную для данной точки.

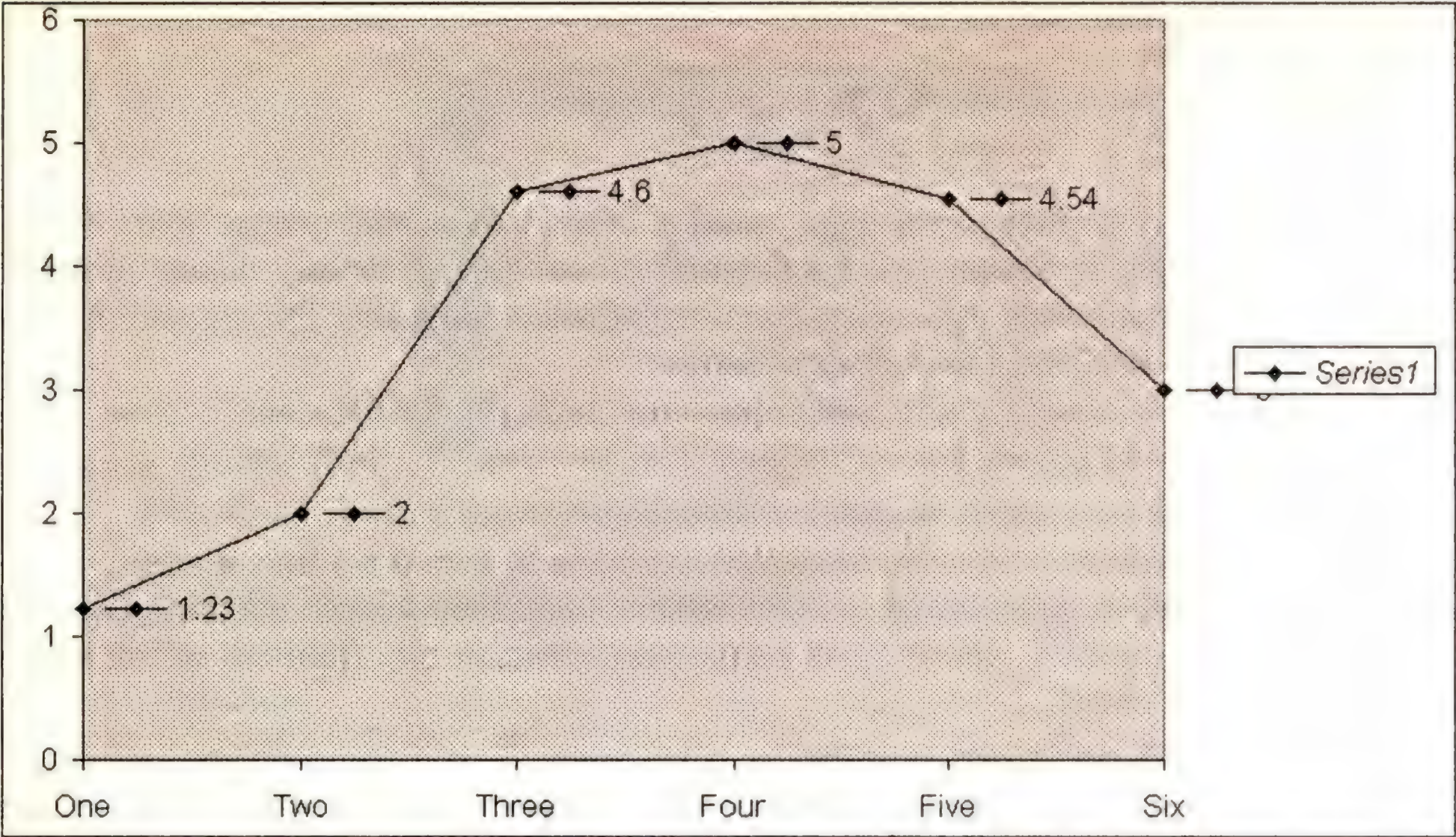


Рис. 21.27. Легенды точек данных

Объект Point

Объект Point представляет собой точку в ряду данных. Для точек одного и того же ряда этот объект содержит семейство Points, являющееся составной частью объекта Series. Это семейство возвращается свойством Points объекта Series.

Объект Point имеет следующих потомков: Border, ChartFillFormat, DataLabel и Interior. Его свойства и методы перечислены в табл. 21.6. Назначение большинства этих свойств понятно без дополнительных пояснений или же объяснялось ранее, поэтому мы подробно остановимся только на двух из них.

Таблица 21.6. Свойства и методы объекта Point

Application	Delete	MarkerSize
ApplyDataLabels	Explosion	MarkerStyle
ApplyPictToEnd	Fill	Parent
ApplyPictToFront	HasDataLabel	Paste
ApplyPictToSides	Interior	PictureType
Border	InvertIfNegative	PictureUnit
ClearFormats	MarkerBackgroundColor	SecondaryPlot
Copy	MarkerBackgroundColorIndex	Select
Creator	MarkerForegroundColor	Shadow
DataLabel	MarkerForegroundColorIndex	

Свойство Explosion

Свойство Explosion возвращает или задает степень разбиения для круговой или кольцевой диаграммы. На рис. 21.28 показана диаграмма со степенью разбиения, равной 20, полученная в результате выполнения следующего кода:

```
ActiveChart.SeriesCollection(1).Points(2).Explosion = 20
```

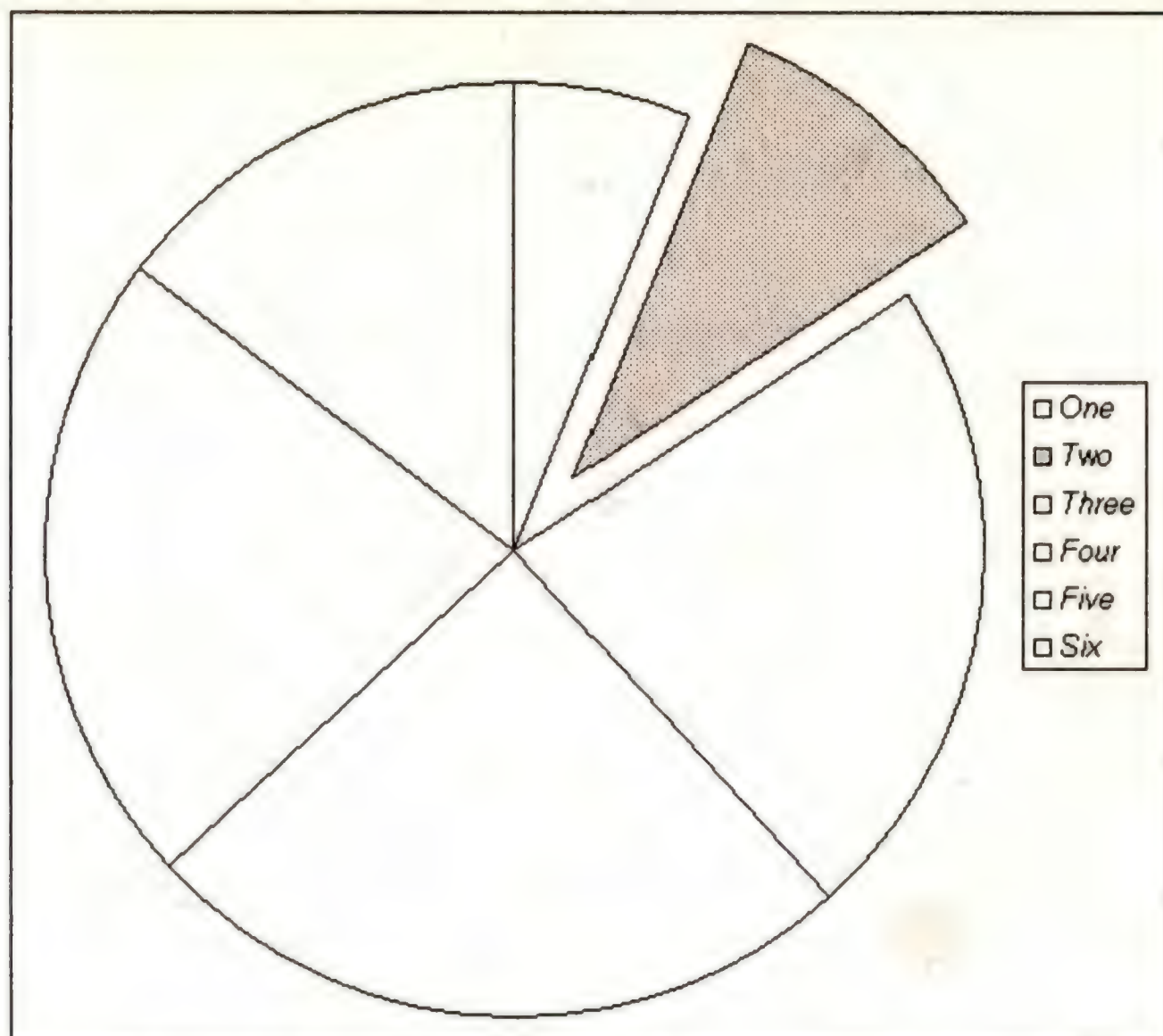



Рис. 21.28. Свойство Explosion имеет значение 20

Свойство Explosion может применяться также к рядам данных. В этом случае разбиению подвергаются все сегменты. Значение 0 соответствует отсутствию разбиения.

Свойства MarkerSize и MarkerStyle

Свойство MarkerSize возвращает или задает размер точки данных в точках (в виде числа формата Long). Это свойство также применимо к объекту Series. При этом размер всех маркеров, входящих в ряд, задается одновременно.

Свойство MarkerStyle определяет стиль точки данных и может принимать значения констант из следующего перечисления:

```
Enum XlMarkerStyle
    xlMarkerStyleX = -4168
    xlMarkerStylePicture = -4147
    xlMarkerStyleNone = -4142
    xlMarkerStyleDot = -4118
    xlMarkerStyleDash = -4115
    xlMarkerStyleAutomatic = -4105
    xlMarkerStyleSquare = 1
    xlMarkerStyleDiamond = 2
    xlMarkerStyleTriangle = 3
    xlMarkerStyleStar = 5
    xlMarkerStyleCircle = 8
    xlMarkerStylePlus = 9
End Enum
```

Для иллюстрации рассмотрим код, формирующий диаграмму, изображенную на рис. 21.29:

```
With ActiveChart.SeriesCollection(1)
    .MarkerSize = 10
```



```
.MarkerStyle = xlMarkerStyleDiamond
With .Points(2)
    .MarkerSize = 20
    .MarkerStyle = xlMarkerStyleCircle
End With
End With
```

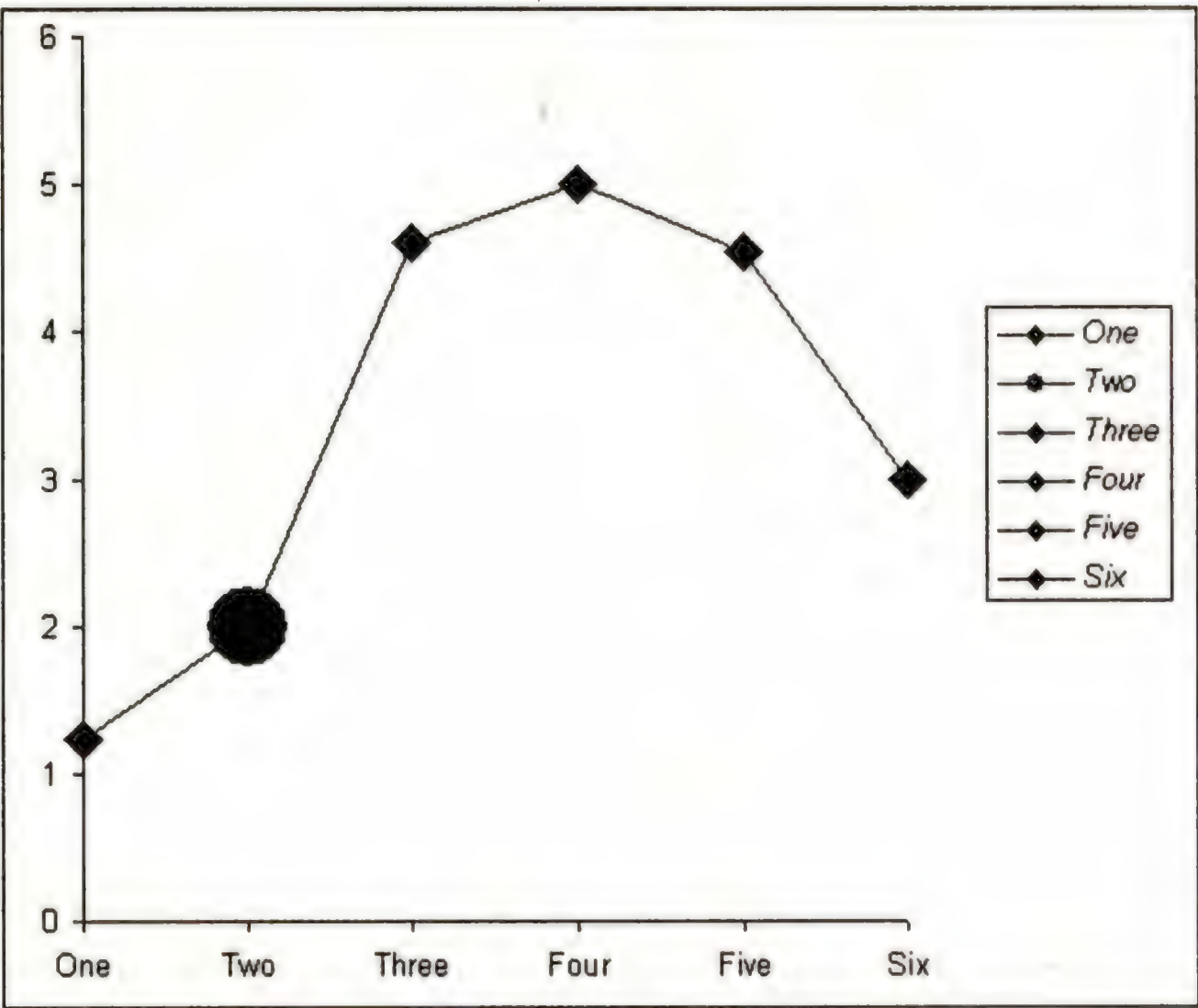


Рис. 21.29. Иллюстрация свойств MarkerSize и MarkerStyle

Свойства и методы объекта Chart

Объект Chart имеет 91 свойство и метод, перечисленные в табл. 21.7.

Таблица 21.7. Свойства и методы объекта Chart

_ApplyDataLabels<v10>	DoughnutGroups	PieGroups
_CodeName	DrawingObjects	PivotLayout<v9>
_Dummy23	Drawings	PlotArea
_Evaluate	DropDowns	PlotBy
_PrintOut<v9>	Elevation	PlotVisibleOnly
_Protect<v10>	Evaluate	Previous
_SaveAs<v10>	Export	PrintOut
Activate	Floor	PrintPreview
Application	GapDepth	Protect
ApplyCustomType	GetChartElement	ProtectContents
ApplyDataLabels	GroupBoxes	ProtectData
Arcs	GroupObjects	ProtectDrawingObjects
Area3Dgroup	HasAxis	ProtectFormatting
AreaGroups	HasDataTable	ProtectGoalSeek

AutoFormat	HasLegend	ProtectionMode
AutoScaling	HasPivotFields<v9>	ProtectSelection
Axes	HasTitle	RadarGroups
Bar3Dgroup	HeightPercent	Rectangles
BarGroups	Hyperlinks	Refresh
BarShape	Index	RightAngleAxes
Buttons	Labels	Rotation
ChartArea	Legend	SaveAs
ChartGroups	Line3DGroup	Scripts<v9>
ChartObjects	LineGroups	ScrollBars
ChartTitle	Lines	Select
ChartType	ListBoxes	SeriesCollection
ChartWizard	Location	SetBackgroundPicture
CheckBoxes	MailEnvelope<v10>	SetSourceData
CheckSpelling	Move	Shapes
CodeName	Name	ShowWindow
Column3Dgroup	Next	SizeWithWindow
ColumnGroups	OLEObjects	Spinners
Copy	OnDoubleClick	SubType
CopyChartBuild	OnSheetActivate	SurfaceGroup
CopyPicture	OnSheetDeactivate	Tab<v10>
Corners	OptionButtons	TextBoxes
CreatePublisher	Ovals	Type
Creator	PageSetup	Unprotect
DataTable	Parent	Visible
Delete	Paste	Walls
DepthPercent	Perspective	WallsAndGridlines2D
Deselect	Pictures	XYGroups
DisplayBlanksAs	Pie3DGroup	

В табл. 21.8 перечислены свойства и методы объекта Chart, возвращающие его потомков вместе с возвращаемыми ими объектами. Помните, что некоторые свойства и методы могут возвращать как один объект, так и их семейство.

Таблица 21.8. Свойства и методы, возвращающие потомков объекта Chart

Название свойства или метода	Тип возвращаемого значения
Application	Application
Area3Dgroup	ChartGroup
AreaGroups	ChartGroup(s)
Axes	Axis/Axes
Bar3Dgroup	ChartGroup
BarGroups	ChartGroup(s)
ChartArea	ChartArea
ChartGroups	ChartGroup(s)

продолжение ➤

Таблица 21.8 (продолжение)

Название свойства или метода	Тип возвращаемого значения
ChartObjects	ChartObject(s)
ChartTitle	ChartTitle
Column3Dgroup	ChartGroup
ColumnGroups	ChartGroup(s)
Corners	Corners
DataTable	DataTable
DoughnutGroups	ChartGroup(s)
Floor	Floor
Hyperlinks	Hyperlinks
Legend	Legend
Line3Dgroup	ChartGroup
Location	Chart
OLEObjects	OLEObject(s)
PageSetup	PageSetup
Pie3Dgroup	ChartGroup
PieGroups	ChartGroup(s)
PlotArea	PlotArea
RadarGroups	ChartGroup(s)
SeriesCollection	Series/SeriesCollection
Shapes	Shapes
SurfaceGroup	ChartGroup
Walls	Walls
XYGroups	ChartGroup(s)

Рассмотрим некоторые свойства и методы объекта Chart. Впрочем, о многих из них уже не раз упоминалось в связи с другими объектами, связанными с диаграммами.

Метод ChartWizard

Метод ChartWizard изменяет свойства диаграммы. К сожалению, при помощи этого метода невозможно создать новую диаграмму, зато он позволяет применить несколько вариантов форматирования одновременно. Метод ChartWizard меняет только свойства, которые определяются параметрами, используемыми при вызове метода.

Обратите внимание на синтаксис данного метода:

```
ChartObject.ChartWizard(Source, Gallery, Format,
    PlotBy, CategoryLabels, SeriesLabels, HasLegend,
    Title, CategoryTitle, ValueTitle, ExtraTitle)
```

Все параметры данного метода являются необязательными.

Параметр *Source* задает диапазон, содержащий источник данных для диаграммы. Пропуск этого параметра приводит к тому, что Excel использует выделенную встроенную диаграмму или активную диаграмму, расположенную на отдельном листе. При отсутствии в книге этих объектов появляется сообщение об ошибке.

Параметр *Gallery* указывает тип диаграммы и может принимать следующие значения: *xlArea*, *xlBar*, *xlColumn*, *xlLine*, *xlPie*, *xlRadar*, *xlXYScatter*, *xlCombination*, *xl3DArea*, *xl3DBar*, *xl3DColumn*, *xl3DLine*, *xl3DPie*, *xl3DSurface*, *xlDoughnut* или *xlDefaultAutoFormat*.

Параметр *Format* указывает подтип диаграммы в соответствии с типом, заданным значением параметра *Gallery*. Параметр *Format* может принимать значения от 1 до 10, в зависимости от типа диаграммы. Эти значения соответствуют типам диаграмм в диалоговом окне Chart Type (Тип диаграммы). Пропуск этого аргумента приводит к тому, что приложение выбирает значение, основываясь на выбранном типе диаграммы и источнике данных.

Параметр *PlotBy* указывает на расположение рядов данных. Он может принимать одно из двух значений *xlRows* или *xlColumns*. В первом случае ряды расположены в виде строк, во втором — в виде столбцов.

Параметр *CategoryLabels* представляет собой целое число, определяющее количество строк или столбцов в диапазоне, содержащем источник данных, которые заняты метками категорий. Этот параметр может принимать любое значение от 0 до $n-1$, где n — максимальное число категорий или рядов.

Соответственно, параметр *SeriesLabels* представляет собой целое число, указывающее количество строк или столбцов в диапазоне, содержащем источник данных, которые заняты метками рядов. Этот параметр может принимать любое значение от 0 до $n-1$, где n — максимальное число категорий или рядов.

Чтобы включить в диаграмму легенду, достаточно присвоить параметру *HasLegend* значение *True*.

Параметр *Title* определяет текст заголовка диаграммы. Соответственно, параметр *CategoryTitle* задает подпись к оси категорий, а параметр *ValueTitle* — подпись к оси значений. Для трехмерных диаграмм существует также параметр *ExtraTitle*, задающий заголовок оси рядов. Он же определяет заголовки вторичных осей на двухмерных диаграммах.

Для иллюстрации рассмотрим диаграмму, изображенную на рис. 21.30. Ниже приводится код, преобразующий эту диаграмму таким образом, что получается объект, отображенный на рис. 21.31:

```
ActiveChart.ChartWizard Gallery:=xlLine, _  
Format:=1, HasLegend:=True, Title:="Averages", _  
CategoryTitle:="Grades", ValueTitle:="Average".
```

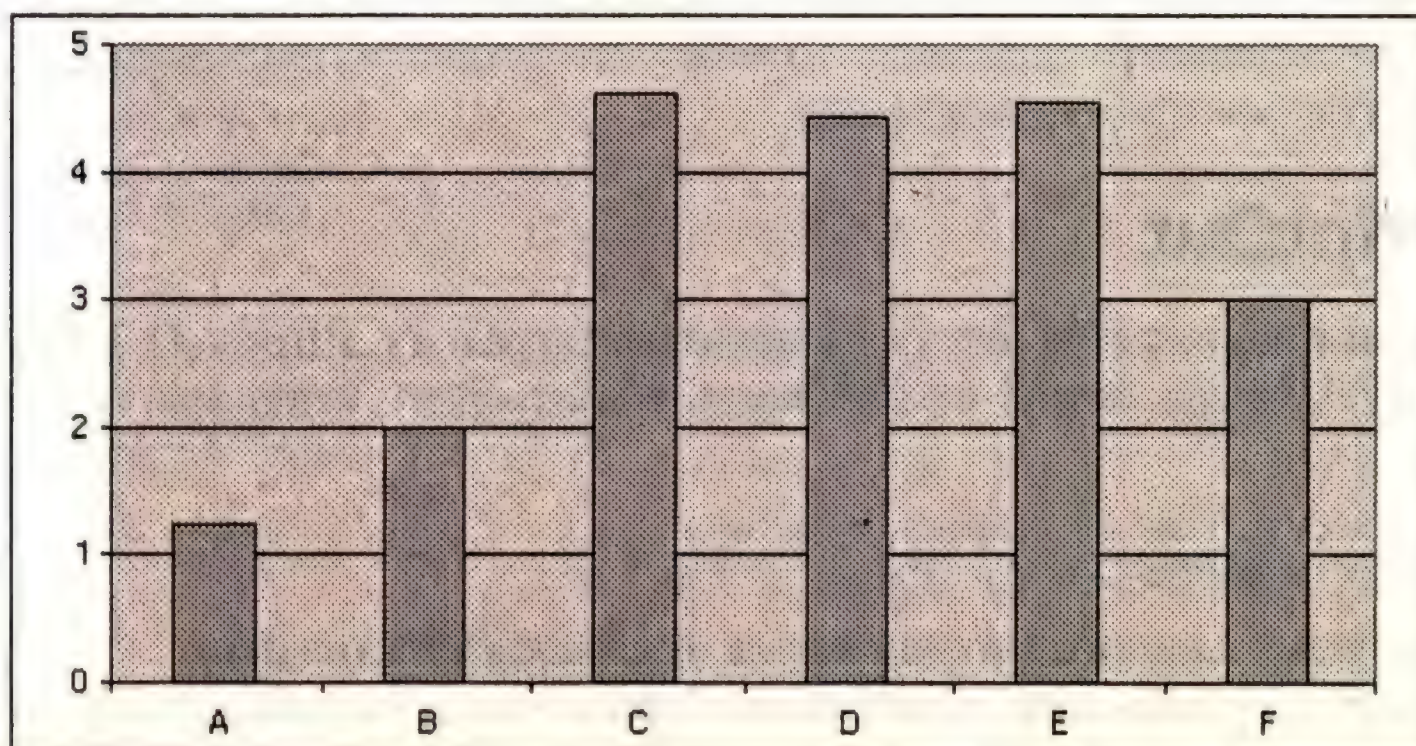


Рис. 21.30. Диаграмма, на основе которой будет показано применение метода ChartWizard

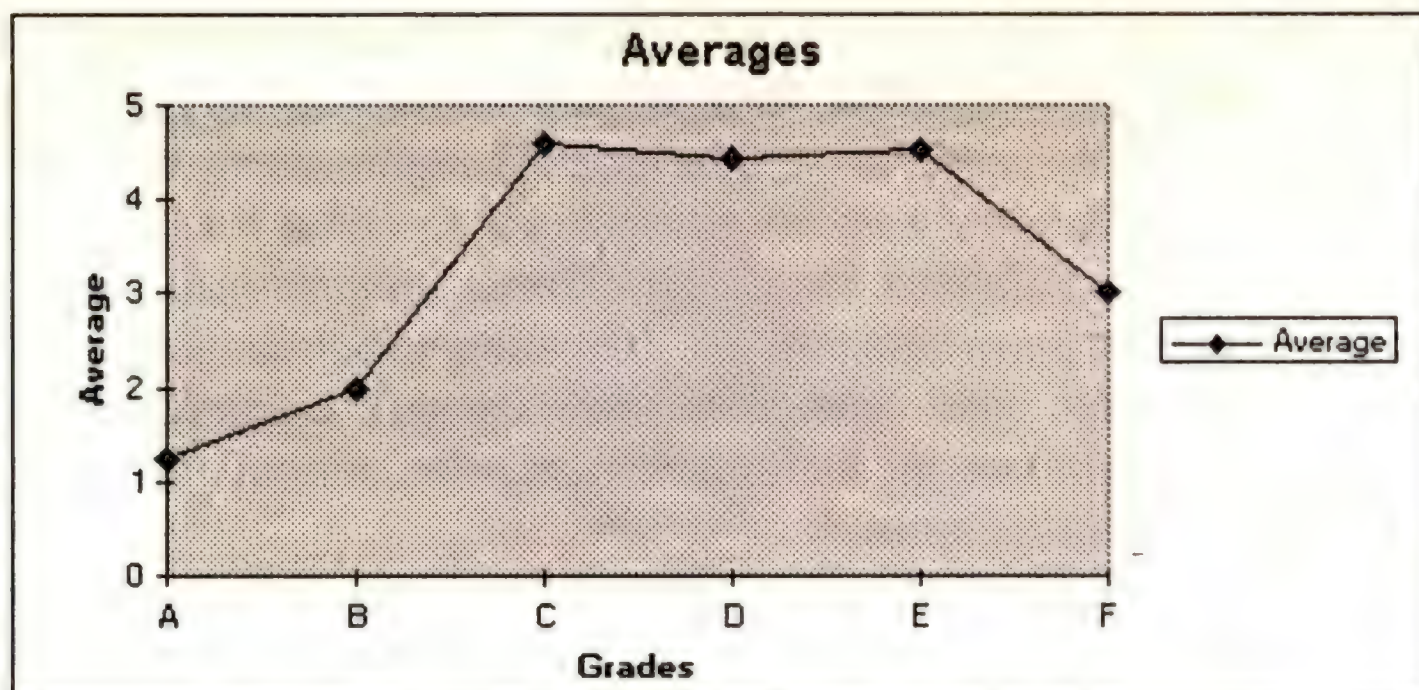


Рис. 21.31. Результат применения метода ChartWizard

Метод Export

Метод Export позволяет перевести диаграмму в графический формат. Вот его синтаксис:

```
ChartObject.Export(FileName, FilterName, Interactive)
```

Здесь параметр *FileName* представляет собой имя создаваемого графического файла, а параметр *FilterName* — имя графического фильтра, определенное в системном реестре. В документации Microsoft не указано, в каком месте системного реестра отображается это имя, но, скорее всего, он будет в следующем ключе:

HKEY_LOCAL_MACHINE\Software\Microsoft\Shared Tools\Graphics Filters\Export

Чтобы отобразить диалоговое окно с параметрами фильтрации, следует присвоить значение True параметру *Interactive*. Если этот аргумент имеет значение False, приложение использует для фильтрации заданное по умолчанию значение.

Для иллюстрации рассмотрим код, создающий на базе активной диаграммы три графических файла различных форматов:

```
ActiveChart.Export "d:\excel\test.png", "PNG"
ActiveChart.Export "d:\excel\test.jpg", "JPEG"
ActiveChart.Export "d:\excel\test.gif", "GIF"
```

Опыт подсказывает, что в данном случае присвоение параметру *Interactive* значения True не оказывает никакого эффекта. Кроме того, имейте в виду, что в случае присваивания параметру *FileName* имени уже существующего файла последний переписывается без предупреждения.

Метод PrintOut

Метод PrintOut позволяет вывести на печать диаграмму. Впрочем, он применим и к ряду других объектов, в том числе Charts, Worksheet(s), Workbook(s), Range. Вот его синтаксис:

```
ChartObject.PrintOut(from, To, Copies, Preview, _
    ActivePrinter, PrintToFile, Collate)
```

Все параметры данного метода являются необязательными.

Параметр *From* указывает номер первой распечатываемой страницы, а параметр *To* — номер последней. Пропуск этих параметров приводит к тому, что выводится на печать весь объект (диапазон, лист и т. п.).

Параметр *Copies* определяет количество распечатываемых копий. По умолчанию он имеет значение 1.

Присвоение параметру *SetPreview* значения True приводит к тому, что становится возможным предварительный просмотр распечатываемого материала. По умолчанию этот параметр имеет значение False.

Параметр *ActivePrinter* определяет имя используемого принтера.

Присвоение параметру *PrintToFile* значения True приводит к тому, что Excel выводит печатаемые данные в файл. При этом пользователь должен указать имя выходного файла. К сожалению, не существует способа задать это имя внутри кода.

Параметр *Collate* должен иметь значение True, чтобы осуществить сортировку множества многостраничных копий.

Пример: прокрутка списка типов диаграмм

В Excel существует 73 различных типа диаграмм. Их слишком много, чтобы вручную проверять вид каждой из них. Однако можно создать приложение, осуществляющее прокрутку типов диаграмм, что дает возможность выбрать наиболее подходящий для конкретного случая тип.

Начать следует с создания диаграммы (разумеется, на основе неких данных, например, как на рис. 21.6) на отдельном листе. Затем в модуль кода листа с диаграммой добавляется код из листинга 21.3.

Листинг 21.3. Код, содержащийся в модуле листа диаграммы

```
Option Explicit
```

```
Dim bPause As Boolean
```

```
Sub ScrollChartTypes()
```

```
Dim iType As Integer, sName As String
```

```
Dim fr As Integer
```

```
fr = FreeFile
```

```
Open ThisWorkbook.Path &
```

```
"\charttypes.txt" For Input As #fr
```

```
Do While Not EOF(fr)
```

```
Input #fr, iType, sName
```

```
On Error Resume Next
```

```
ActiveChart.ChartType = iType
```

```
ActiveChart.HasTitle = True
```

```
ActiveChart.ChartTitle.Text = _  
iType & " -- " & sName
```

```
Delay 2
```

```
If bPause Then
```

```
Do
```

```
DoEvents
```

```
Loop Until bPause = False
```

```
End If
```

```
Loop
```

```
Close fr
```

```
End Sub
```

```
'-----
```

```
Sub Delay(rTime As Single)
```

продолжение ➤

Листинг 21.3 (продолжение)

```
' Временная задержка rTime в секундах (min=.01, max=300)
Dim OldTime As Variant
'Safty net
If rTime < 0.01 Or rTime > 300 Then rTime = 1
OldTime = Timer
Do
    DoEvents
Loop Until Timer - OldTime >= rTime
End Sub

'-----
```

```
Private Sub Chart_MouseDown(ByVal Button As Long, ByVal Shift As Long, ByVal X As Long,
ByVal Y As Long)
    If Button = xlPrimaryButton Then _
        bPause = Not bPause
End Sub
```

Этот код содержит три процедуры. Основная процедура `ScrollChartTypes` определяет тип диаграммы и в соответствии с этим редактирует ее заголовок. Она использует текстовый файл `ChartTypes.txt`, который содержит список всех типов диаграмм с их именами. Содержимое этого текстового файла показано в листинге 21.4. Он должен располагаться в той же папке, что и активная книга.

Листинг 21.4. Файл ChartTypes.txt

```
-4169    XYScatter
-4151    Radar
-4120    Doughnut
-4102    3DPie
-4101    3DLine
-4100    3DColumn
-4098    3DArea
    1    Area
    4    Line
    5    Pie
    15    Bubble
    51    ColumnClustered
    52    ColumnStacked
    53    ColumnStacked100
    54    3DColumnClustered
    55    3DColumnStacked
    56    3DColumnStacked100
    57    BarClustered
    58    BarStacked
    59    BarStacked100
    60    3DBarClustered
    61    3DBarStacked
    62    3DBarStacked100
    63    LineStacked
    64    LineStacked100
    65    LineMarkers
    66    LineMarkersStacked
    67    LineMarkersStacked100
    68    PieOfPie
    69    PieExploded
    70    3DPieExploded
    71    BarOfPie
    72    XYScatterSmooth
    73    XYScatterSmoothNoMarkers
```



```
74 XYScatterLines
75 XYScatterLinesNoMarkers
76 AreaStacked
77 AreaStacked100
78 3DAreaStacked
79 3DAreaStacked100
80 DoughnutExploded
81 RadarMarkers
82 RadarFilled
83 Surface
84 SurfaceWireframe
85 SurfaceTopView
86 SurfaceTopViewWireframe
87 Bubble3DEffect
88 StockHLC
89 StockOHLC
90 StockVHLC
91 StockVOHLC
92 CylinderColClustered
93 CylinderColStacked
94 CylinderColStacked100
95 CylinderBarClustered
96 CylinderBarStacked
97 CylinderBarStacked100
98 CylinderCol
99 ConeColClustered
100 ConeColStacked
101 ConeColStacked100
102 ConeBarClustered
103 ConeBarStacked
104 ConeBarStacked100
105 ConeCol
106 PyramidColClustered
107 PyramidColStacked
108 PyramidColStacked100
109 PyramidBarClustered
110 PyramidBarStacked
111 PyramidBarStacked100
112 PyramidCol
```

Обратите внимание на строку On Error в процедуре ScrollChartTypes, которая возобновляет выполнение процедуры в том случае, если пользователь пытается выбрать тип, не подходящий для рассматриваемой диаграммы.

Процедура Delay просто переводит приложение в режим ожидания на определенное число секунд. Ну и наконец, событиеMouseDown изменяет состояние логической переменной уровня модуля bPause. Щелчок левой кнопкой мыши вызывает паузу в прокрутке типов диаграмм. Для повторного запуска требуется еще раз щелкнуть левой кнопкой мыши. Чтобы полностью остановить процедуру, достаточно нажать комбинацию клавиш Ctrl + Break.

Пример: вывод на печать встроенной диаграммы

Теперь можно добавить в наше приложение SRXUtils функцию PrintCharts. Она формирует список встроенных диаграмм, существующих в активной книге, и дает

пользователю возможность выделить нужные и распечатать их. Для вывода на печать листов диаграмм используется служебная программа PrintSheets.

Интеграция служебной программы PrintCharts ничем не отличается от включения служебных программ PrintSheets и PrintPivotTables, описанной в предыдущих главах книги. В настоящее время служебная программа печати, относящаяся к надстройке Print.utl, просто отображает информационное окно. В функционирующем состоянии эта программа приводит к отображению диалогового окна, показанного на рис. 21.32.

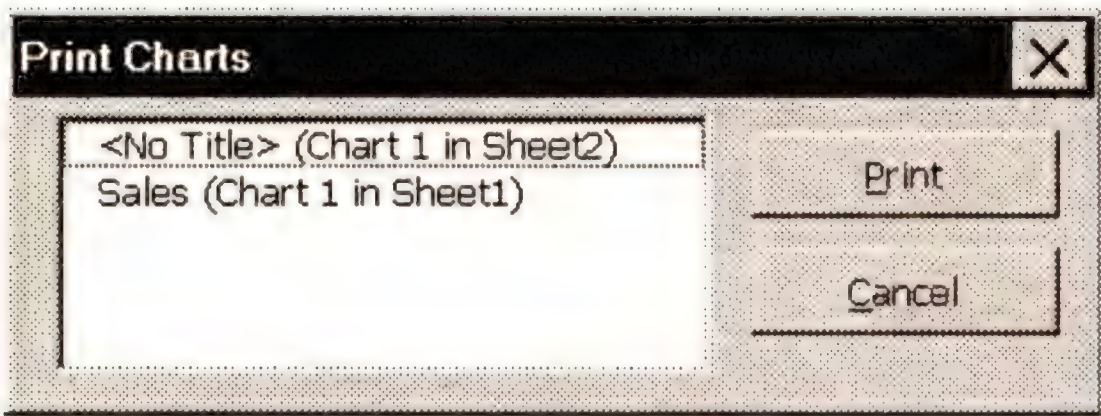


Рис. 21.32. Диалоговое окно Print Charts

Поле со списком содержит перечень всех встроенных диаграмм, находящихся в активной книге. Пользователю достаточно выделить один или несколько элементов списка и нажать кнопку Print.

Перечислим основные этапы создания служебной программы. Так как данный процесс осуществляется в книге Print.xls, для начала требуется открыть эту книгу. После внесения изменений потребуется также сохранить эту книгу под именем Print.utl. Если надстройка Print.utl уже загружена, единственным способом выгрузить ее является снятие флажка SRXUtils.xla и закрытие книги SRXUtils.xls.

Создание объекта UserForm

Создайте в книге Print.xls диалоговое окно, показанное на рис. 21.32, и присвойте ему имя dlgPrintCharts. Свойству Caption присвойте значение Print Charts и создайте процедуру PrintCharts:

```
Public Sub PrintCharts()  
    dlgPrintCharts.Show  
End Sub
```

Диалоговое окно dlgPrintCharts имеет две командные кнопки и поле со списком.

Поле со списком

Поместите поле со списком в форму, как показано на рис. 21.32. В диалоговом окне Properties установите свойства, перечисленные в табл. 21.9:

Таблица 21.9. Свойства поля со списком

Свойство	Значение
Name	IstCharts
TabIndex	0
MultiSelect	frmMultiSelectExtended

Свойство `MultiSelect` имеет значение `frmMultiSelectExtended`, то есть пользователь с помощью клавиши `Ctrl` может выделить несколько непоследовательных имен листов, а с помощью клавиши `Shift` — набор последовательных имен листов.

Свойство `TabIndex` указывает не только на последовательность активизации элементов формы при нажатии клавиши `Tab`, но и элемент, который будет активен при первой загрузке формы. Так как изначально активным должно быть поле со списком, свойству `TabIndex` этого элемента присваивается значение `0`.

Кнопка `Print`

Поместите командную кнопку в форму, изображенную на рис. 21.32, и в окне `Properties` задайте свойства, перечисленные в табл. 21.10:

Таблица 21.10. Свойства кнопки `Print`

Свойство	Значение
<code>Name</code>	<code>cmdPrint</code>
<code>Accelerator</code>	<code>P</code>
<code>Caption</code>	<code>Print</code>
<code>TabIndex</code>	<code>1</code>

Кнопка `Cancel`

Поместите еще одну командную кнопку в форму, изображенную на рис. 21.32, и в окне `Properties` задайте следующие свойства, перечисленные в табл. 21.11:

Таблица 21.11. Свойства кнопки `Cancel`

Свойство	Значение
<code>Name</code>	<code>cmdCancel</code>
<code>Accelerator</code>	<code>C</code>
<code>Caption</code>	<code>Cancel</code>
<code>TabIndex</code>	<code>2</code>
<code>Cancel</code>	<code>True</code>

Когда свойству `Cancel` кнопки `cmdCancel` присвоено значение `True`, срабатывание кнопки происходит, когда пользователь нажимает клавишу `Esc`. То есть нажатие этой клавиши будет приводить к исчезновению диалогового окна.

Создание кода для объекта `UserForm`

Пришло время сопоставить с элементами управления некий код.

Область описаний

Область описаний должна содержать описание переменных уровня модуля, как показано в листинге 21.5.

Листинг 21.5. Описание переменных уровня модуля

```
Option Explicit
Dim cCharts As Integer
Dim sChartObjNames() As String
Dim sSheets() As String
```


Код кнопки Cancel

Код, соответствующий кнопке Cancel, показан в листинге 21.6.

Листинг 21.6. Обработка события Click для кнопки Cancel

```
Private Sub cmdCancel_Click()  
    Unload Me  
End Sub
```

Код кнопки Print

Нажатие кнопки Print вызывает основную процедуру и затем выгружает форму; код соответствующего события содержится в листинге 21.7.

Листинг 21.7. Процедура cmdPrint_Click

```
Private Sub cmdPrint_Click()  
    PrintSelectedCharts  
    Unload Me  
End Sub
```

Событие Initialize формы

Событие Initialize объекта UserForm должно приводить к заполнению поля со списком — перечнем встроенных диаграмм. Наше приложение использует один массив уровня модуля для сохранения имен диаграмм, второй — для сохранения имен объектов ChartObject и целую переменную уровня модуля для сохранения числа диаграмм. Заполнение этих массивов будет связано с событием Initialize, как показано в листинге 21.8. Затем на основе массивов сформируется список диаграмм. Массивы снова будут использоваться в основной процедуре распечатки. Именно поэтому при описании было указано, что они находятся на уровне модуля.

Обратите внимание на применение оператора ReDim для динамического определения количества элементов в массивах. Это требуется, так как количество встроенных диаграмм в книге заранее неизвестно.

Листинг 21.8. Процедура события Initialize

```
Private Sub UserForm_Initialize()  
  
    ' Заполнение поля со списком — перечнем встроенных диаграмм  
  
    Dim ws As Worksheet  
    Dim chObj As ChartObject  
    ReDim sChartObjNames(1 To 10) As String  
    ReDim sSheets(1 To 10) As String  
  
    lstCharts.Clear  
    cCharts = 0  
    For Each ws In ActiveWorkbook.Worksheets  
        For Each chObj In ws.ChartObjects  
            ' Обновление числа встроенных диаграмм  
            cCharts = cCharts + 1  
            ' При необходимости количество элементов массива меняется  
            If UBound(sSheets) < cCharts Then
```



```
        ReDim Preserve sSheets(1 To cCharts + 5)
        ReDim Preserve sChartObjNames(1 To cCharts + 5)
    End If
    ' Сохранение имени диаграммы
    sChartObjNames(cCharts) = chObj.Name
    sSheets(cCharts) = ws.Name
    ' Добавление элемента в поле со списком
    If chObj.Chart.HasTitle Then
        lstCharts.AddItem chObj.Chart.ChartTitle.Text & " (" & _
            sChartObjNames(cCharts) & " in " & sSheets(cCharts) & ")"
    Else
        lstCharts.AddItem "<No Title> (" & sChartObjNames(cCharts) & " in " & _
            sSheets(cCharts) & ")"
    End If
Next
End Sub
```

Процедура PrintCharts

Основная процедура вывода на печать показана в листинге 21.9. Особое внимание требуется обратить на два момента. Во-первых, в книге может вообще не содержаться встроенных диаграмм. Во-вторых, пользователь может нажать кнопку Print, не выделив ни одного элемента в поле со списком.

Важно понимать, что все поля со списком представляют собой массивы, нумерация которых начинается с нуля, а нумерация нашего массива со списком встроенных диаграмм начинается с единицы. То есть следует помнить, что члену массива номер i соответствует индекс $i + 1$.

Листинг 21.9. Процедура PrintSelectedCharts

```
Sub PrintSelectedCharts()
    ' Вывод на печать диаграмм, выделенных в поле со списком
    Dim i As Integer
    Dim bNoneSelected As Boolean

    bNoneSelected = True

    If cCharts = 0 Then
        MsgBox "В книге отсутствуют встроенные диаграммы.", vbExclamation
        Exit Sub
    Else
        For i = 0 To lstCharts.ListCount - 1
            If lstCharts.Selected(i) Then
                bNoneSelected = False
                ' Нумерация в поле со списком начинается с 0, а нумерация массива с 1
                Worksheets(sSheets(i + 1))._
                    ChartObjects(sChartObjNames(i + 1)).Chart.PrintOut
            End If
        Next
    End If

    If bNoneSelected Then
        MsgBox "Не было выделено ни одного имени диаграммы.", vbExclamation
    End If
End Sub
```


Пример: установка меток рядов данных

Может быть, вы уже знаете, что для получения возможности поменять подписи к данным, требуется щелкнуть на них два раза. Это переводит подпись в режим редактирования, как показано на рис. 21.33.

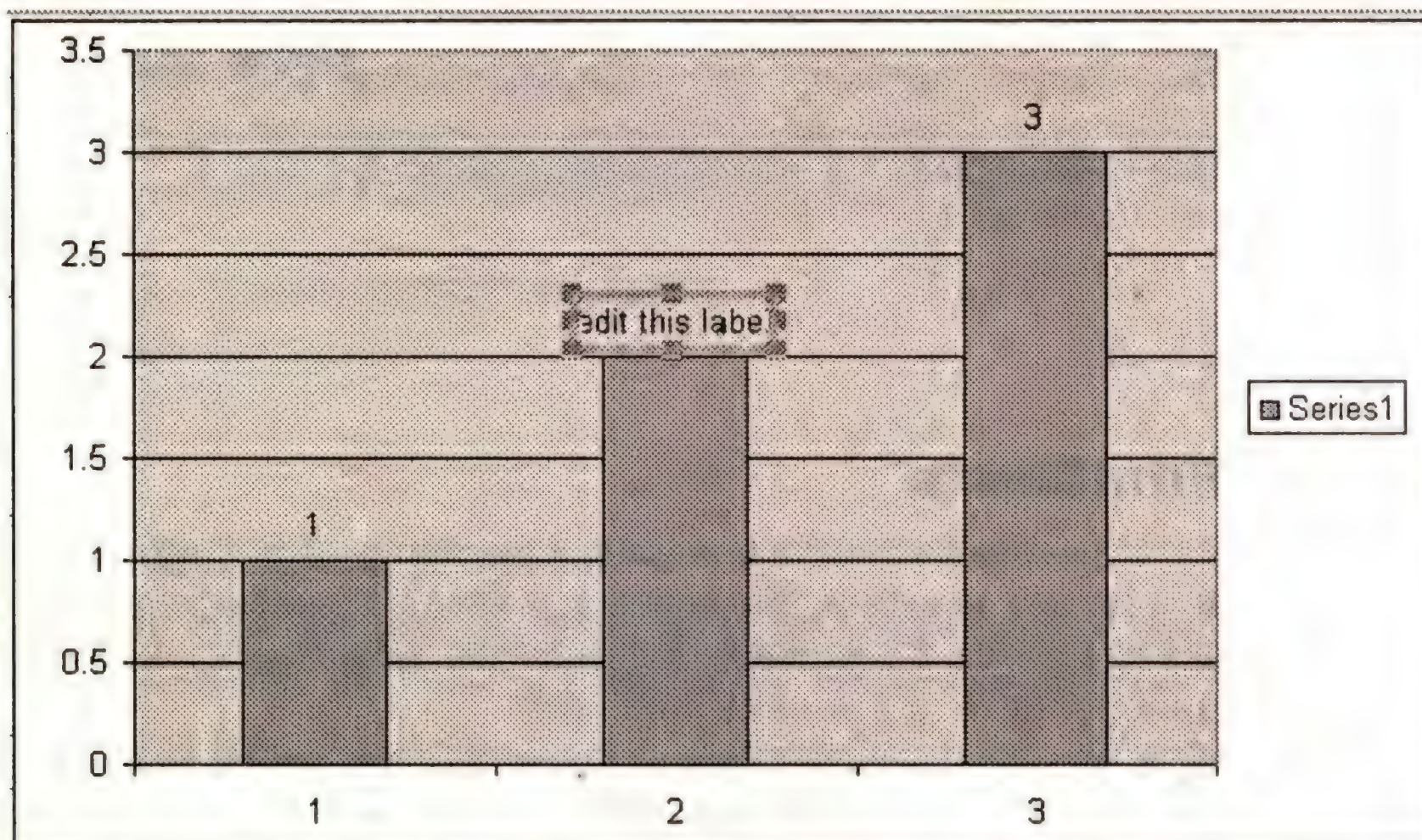


Рис. 21.33. Подпись к данным в режиме редактирования

В режиме редактирования можно изменять текст подписи данных (разрывая все связи), а также создавать ссылку на новую ячейку. Например, код:

```
ActiveChart.SeriesCollection(1).DataLabels(2).Text = "=MyChartSheet!R12C2"
```

задает подпись данных для второй точки в виде значения ячейки B12. Обратите внимание: формула должна определяться в стиле R1C1. Но, к сожалению, в Excel не существует простого способа быстро связать все подписи к данным для рядов с диапазоном листа. Остается только вручную обрабатывать подписи данных по очереди. Напишем служебную программу для этой цели и добавим ее к надстройке SRXUtils.

При вызове этой служебной программы появляется диалоговое окно, показанное на рис. 21.34 (со списком всех рядов данных для выделенной диаграммы). Пользователь может выбрать ряд данных и затем определить как диапазон ячеек, с которыми будут связаны подписи, так и диапазон ячеек, чьи значения копируются в эти подписи. В последнем случае связь между значениями ячеек и подписями не устанавливается и, соответственно, изменения диапазона значений этих ячеек не отображаются на диаграмме. В зависимости от желания пользователя форматирование может быть как связано с форматированием ячейки, так и просто скопировано.

Начнем с пополнения листа DataSheet, к которому будет добавлена строка, соответствующая новой служебной программе, как показано на рис. 21.35 (новая программа фигурирует в строке 2).

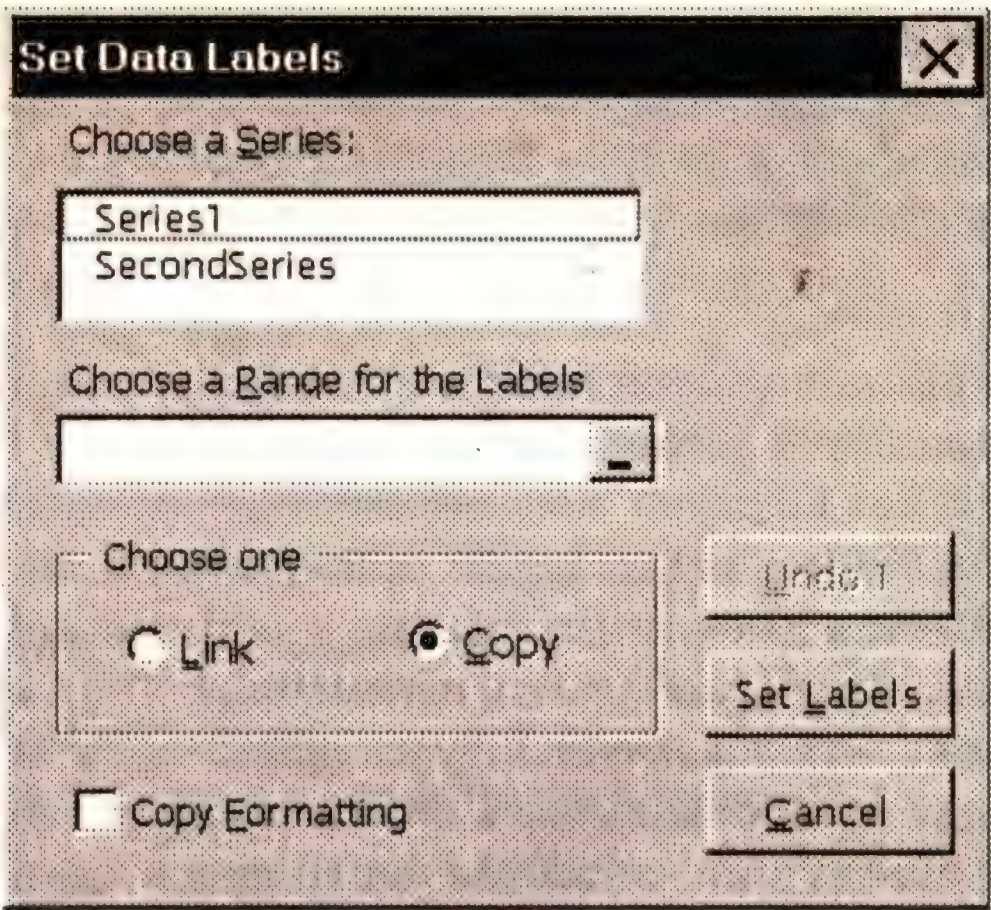


Рис. 21.34. Диалоговое окно Set Data Labels

	A	B	C	D	E	F	G	H
1	Utility	OnAction Proc	Procedure	In Workbook	Menu Item	SubMenu Item	On Wks Menu	On Chart Menu
2	Activate Sheet	RunUtility	ActivateSheet	ThisWorkbook	&Activate Sheet		TRUE	TRUE
3	Label Data Series	RunUtility	LabelDataSeries	ThisWorkbook	&Label Data Series		TRUE	TRUE
4	Print Charts	RunUtility	PrintCharts	Print.utl	&Print	Embedded &Charts	TRUE	TRUE
5	Print Pivot Tables	RunUtility	PrintPivotTables	Print.utl		&Pivot Tables	TRUE	TRUE
6	Print Sheets	RunUtility	PrintSheets	Print.utl		&Sheets	TRUE	TRUE
7	Select Special	RunUtility	SelectSpecial	ThisWorkbook	S&elect Special		TRUE	FALSE
8	Sort Sheets	RunUtility	SortSheets	ThisWorkbook	&Sort Sheets		TRUE	TRUE

Рис. 21.35. Лист DataSheet

Затем нужно создать диалоговое окно, показанное на рис. 21.34. В данном случае использовался элемент управления RefEdit, имитирующий выделение диапазона с помощью мыши. К сожалению, в справочной документации отсутствуют сведения об этом элементе. Можно получить список его свойств, методов и событий в браузере объектов Microsoft Object Browser, но не более. В любом случае нам потребуется только одно или два свойства, как станет ясно из кода.

Процедуру LabelDataSeries, вызываемую при выделении элемента меню, следует поместить в новый стандартный модуль кода с именем basDataLabels. Описания переменных этого модуля даны в листинге 21.10.

Листинг 21.10. Область описания модуля basDataLabels

```
Option Explicit

Private Type utDataLabels
    HasDataLabel As Boolean
    Label As String
    FontName As String
    FontSize As Single
    Color As Long
    Bold As Boolean
    Italic As Boolean
End Type

Public LabelsForUndo() As utDataLabels
```

продолжение ↗

Листинг 21.10 (продолжение)

```
Public DataSeries As Series
Public cPoints As Integer
Public bCopyFormatting As Boolean
Public oChart As Chart
Dim cSeries as Long
```

Обратите внимание на пользовательское описание переменной Type. Оно сохраняет исходные подписи к данным для операции Undo. Эта переменная может сохранять не только текст подписи, но и параметры форматирования.

Процедура LabelDataSeries, код которой приводится в листинге 21.11, сначала проверяет факт выделения именно диаграммы — встроенной или же расположенной на отдельном листе. Затем глобальной переменной oChart присваивается ссылка на активную диаграмму. Эта переменная будет использоваться диалоговым окном Set Data Labels, поэтому она должна быть описана на уровне проекта. Затем глобальной переменной cSeries присваивается число рядов данных в диаграмме. При отсутствии рядов данных появляется соответствующее сообщение. Если все условия соблюдены, отображается диалоговое окно Set Data Labels.

Листинг 21.11. Процедура LabelDataSeries

```
Public Sub LabelDataSeries()

' Проверка активности именно
' встроенной или расположенной на отдельном листе диаграммы.
' Если это так, задается переменная oChart.

Set oChart = Nothing

If ActiveWorkbook.Sheets.Count = 0 Then
    MsgBox "В активной книге нет диаграмм.", vbInformation
    Exit Sub
End If

On Error GoTo NoChartActive
Set oChart = ActiveChart
If Not oChart Is Nothing Then
    ' Проверка наличия рядов данных
    cSeries = oChart.SeriesCollection.Count
    If cSeries = 0 Then
        MsgBox "Активная диаграмма не имеет рядов данных.", vbInformation
        Exit Sub
    End If
    ' Так как все условия соблюдены, отображается диалоговое окно Set Data Labels
    dlgDataLabels.Show
Else
    MsgBox "Выделите диаграмму для запуска приложения.", vbInformation
End If

Exit Sub

NoChartActive:
    MsgBox "Выделите диаграмму для запуска приложения.", vbInformation
Exit Sub

End Sub
```


После создания модуля `basDataLabels` требуется смоделировать диалоговое окно, показанное на рис. 21.34, присвоить ему имя `dlgDataLabels` и сопоставить строку `Set Data Labels` с его свойством `Caption`. Процесс создания диалоговых окон уже описан в предыдущих главах, поэтому в данном случае мы не будем останавливаться на нем подробно. При желании вы можете изменить вид этого диалогового окна. Вот перечень элементов управления:

- `CmdCancel` для кнопки `Cancel`;
- `CmdSetLabels` для кнопки `Set Labels`;
- `cmdUndo` для кнопки `Undo`;
- `LblSeries` для надписи `Choose a Series::`;
- `LstSeries` для поля со списком;
- `LblRange` для надписи `Choose a Range for the Labels`;
- `reditLabels` для элемента управления `RefEdit`;
- `fraOptions` для рамки;
- `OptLink` для переключателя `Link`;
- `OptCopy` для переключателя `Copy`;
- `chkOption` для флажка `Copy Formatting`;

Думаю, для вас не составит труда понять, каким элементам управления соответствуют эти имена. Пояснить в данном случае требуется только назначение элемента `RefEdit`. Он позволяет выделять диапазон на рабочем листе при открытом модальном окне. У правой границы этого элемента находится кнопка, щелчок на которой на время скрывает диалоговое окно и демонстрирует пользователю лист целиком.

Большинство фрагментов кода, соответствующего элементам управления в диалоговом окне, достаточно очевидны. Событие `Initialize`, код которого показан в листинге 21.12, сначала заполняет поле со списком (`lstSeries`) перечнем рядов данных диаграммы `oChart`, а затем передает инициативу другим элементам управления.

Листинг 21.12. Процедура события `Initialize`

```
Private Sub UserForm_Initialize()  
  
    ' Переменная oChart соответствует активной диаграмме  
    ' Переменная cSeries определяет число рядов данных  
  
    Dim ds As Series  
  
    ' Заполнение поля со списком  
    For Each ds In oChart.SeriesCollection  
        lstSeries.AddItem ds.Name  
    Next  
  
    optCopy.Value = True  
    chkOption.Caption = "Copy Formatting"  
    chkOption.Accelerator = "F"  
    cmdUndo.Enabled = False  
  
End Sub
```


Требуется, чтобы надпись рядом с флажком (а заодно и значение свойства `Accelerator`) зависела от положения расположенного выше переключателя (`Link` или `Copy`). Соответственно, нужно написать код для обоих событий `Click`. Он приведен в листинге 21.13.

Листинг 21.13. События `Click` для положения переключателя

```
Private Sub optCopy_Click()
```

```
    ' Изменение надписи рядом с флажком и значения свойства Accelerator
    chkOption.Caption = "Copy Formatting"
    chkOption.Accelerator = "F"
    chkOption.Enabled = True
```

```
End Sub
```

```
Private Sub optLink_Click()
```

```
    ' Изменение надписи рядом с флажком и значения свойства Accelerator
    chkOption.Caption = "Link Number Format"
    chkOption.Accelerator = "N"
    chkOption.Enabled = True
```

```
End Sub
```

Процедуры события `Click` для командных кнопок обычно имеют небольшой размер. Они приведены в листинге 21.14.

Листинг 21.14. События `Click` для кнопок `cmdCancel` и `cmdSetLabels`

```
Private Sub cmdCancel_Click()
```

```
    Unload Me
```

```
End Sub
```

```
Private Sub cmdSetLabels_Click()
```

```
    DoDataLabels
```

```
End Sub
```

Основная часть кода процедуры `DoDataLabels` показана в листинге 21.15. Она должна располагаться в модуле объекта `UserForm`. Эта процедура проверяет, выделены ли ряды данных и диапазоны подписей, и сравнивает их размеры. При несовпадении размеров появляется сообщение об ошибке.

Листинг 21.15. Процедура `DoDataLabels`

```
Sub DoDataLabels()
```

```
    Dim i As Integer
```

```
    Dim rngLabels As Range
```

```
    Dim fnt As Font
```

```
    ' Выделен ли ряд данных? Зафиксируем его размер.
```

```
    If lstSeries.ListIndex = -1 Then
```

```
        MsgBox "Выделите ряд данных.", vbInformation
```

```
        Exit Sub
```

```
    Else
```

```
        Set DataSeries = oChart.SeriesCollection(lstSeries.ListIndex + 1)
```

```
        ' Если диаграмма не поддерживает точки данных, появится сообщение
        ' об ошибке
```



```
On Error Resume Next
cPoints = DataSeries.Points.Count
If Err.Number <> 0 Then
    MsgBox "Выделенная диаграмма не поддерживает точки данных.", _
        vbCritical
    Unload Me
    Exit Sub
End If

' Определение диапазона подписей
Set rngLabels = Range(reditLabels.Value)
If rngLabels Is Nothing Then
    MsgBox "Выделите диапазон ячеек, число которых равно " & _
        ", то есть числу точек данных в ряду.", vbInformation
    Exit Sub
End If

' Проверка количества подписей
If cPoints <> rngLabels.Count Then
    MsgBox "Число ячеек с подписями (" & rngLabels.Count & _
        ") не равно числу точек данных (" & cPoints & _
        ") в выделенном ряду.", vbInformation
    Exit Sub
End If

' Проверка существующих подписей и их сохранение
If DataSeries.HasDataLabels Then
    ' Определение размерности массива
    ReDim LabelsForUndo(1 To cPoints)
    ' Заполнение массива
    For i = 1 To cPoints
        LabelsForUndo(i).HasDataLabel = DataSeries.Points(i).HasDataLabel
        If LabelsForUndo(i).HasDataLabel Then
            ' Сохранение текста подписи
            LabelsForUndo(i).Label = DataSeries.Points(i).DataLabel.Text
            ' Сохранение форматирования
            With DataSeries.Points(i).DataLabel.Font
                LabelsForUndo(i).FontName = .Name
                LabelsForUndo(i).FontSize = .Size
                LabelsForUndo(i).Color = .Color
                LabelsForUndo(i).Bold = .Bold
                LabelsForUndo(i).Italic = .Italic
            End With
        End If
    Next
    cmdUndo.Enabled = True
End If

' Подписи к данным должны зависеть от параметров
If optLink Then
    For i = 1 To cPoints
        DataSeries.Points(i).HasDataLabel = True
        DataSeries.Points(i).DataLabel.Text = "=" & rngLabels.Parent.Name _
            & "!" & rngLabels.Cells(i).Address(ReferenceStyle:=xlR1C1)
        If chkOption Then
            ' Задание числового формата ссылки
            DataSeries.Points(i).DataLabel.NumberFormatLinked = True
        End If
    Next
Else
```

продолжение ↗

Листинг 21.15 (продолжение)

```

For i = 1 To cPoints
    DataSet.Points(i).HasDataLabel = True
    DataSet.Points(i).DataLabel.Text = rngLabels.Cells(i).Value
    If chkOption Then
        bCopyFormatting = True
        With DataSet.Points(i).DataLabel.Font
            .Name = rngLabels.Cells(i).Font.Name
            .Size = rngLabels.Cells(i).Font.Size
            .Bold = rngLabels.Cells(i).Font.Bold
            .Italic = rngLabels.Cells(i).Font.Italic
            .Color = rngLabels.Cells(i).Font.Color
        End With
        DataSet.Points(i).DataLabel.NumberFormat = _
            rngLabels.Cells(i).NumberFormat
    Else
        bCopyFormatting = False
    End If
Next
End If

End Sub

```

Код события Click командной кнопки Undo приводится в листинге 21.16. Он восстанавливает исходные подписи к данным, которые были сохранены с помощью процедуры DoDataLabels.

Листинг 21.16. Процедура cmdUndo_Click

```

Private Sub cmdUndo_Click()

    ' Восстановление подписей для рядов данных

    Dim i As Integer

    For i = 1 To cPoints
        If LabelsForUndo(i).HasDataLabel Then
            DataSet.Points(i).HasDataLabel = True
            DataSet.Points(i).DataLabel.Text = LabelsForUndo(i).Label
            If bCopyFormatting Then
                ' Восстановление форматирования
                With DataSet.Points(i).DataLabel.Font
                    .Name = LabelsForUndo(i).FontName
                    .Size = LabelsForUndo(i).FontSize
                    .Color = LabelsForUndo(i).Color
                    .Bold = LabelsForUndo(i).Bold
                    .Italic = LabelsForUndo(i).Italic
                End With
            End If
        Else
            DataSet.Points(i).HasDataLabel = False
        End If
    Next
    cmdUndo.Enabled = False

End Sub

```


22

Смарт-теги

В этой главе мы поговорим о программном управлении смарт-тегами. Для создания пользовательских смарт-тегов требуется приложение, позволяющее создавать сервер автоматизации (COM DLL), например Visual Basic или Visual C++. В приложении Excel XP невозможно создать пользовательские смарт-теги, хотя и существует возможность программного контроля уже имеющихся элементов.

Что такое смарт-теги?

Так как в пакете Office смарт-теги появились начиная с версии Office XP, дадим краткое описание их назначения. Представим, что пользователь печатает дату, например 1 января 2002, в приложении Word XP (при включенных смарт-тегах). Система смарт-тегов распознает текст "1 января 2002" в качестве даты и подчеркнет его пунктирной линией. Если затем поместить над датой указатель мыши, появится небольшая кнопка, выглядящая, как вписанная в круг буква "i". Щелчок на этой кнопке приведет к появлению раскрывающегося списка, показанного на рис. 22.1.

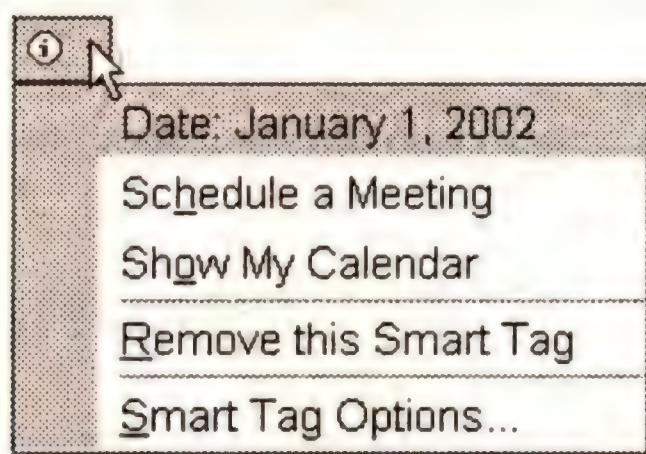


Рис. 22.1. Меню смарт-тега

Выбором команд этого меню можно осуществить определенные действия, например назначить встречу в выбранный день. Для этого достаточно щелкнуть на пункте *Schedule a Meeting* (Назначить встречу). Появится соответствующее диалоговое окно приложения Microsoft Outlook, в котором можно указать параметры встречи.

На рис. 22.2 иллюстрируется использование смарт-тегов в Excel XP. В ячейку было введено сокращенное обозначение акций, после чего это обозначение было распознано как смарт-тег. В Excel XP существует механизм встроенного распозна-

вания смарт-тегов. Его действие проявляется на вкладке Smart-tag (Смарт-теги) диалогового окна AutoCorrect Options (Параметры автозамены).

Таким образом, смарт-теги являются инструментами, которые распознают определенные типы данных в документах Office и дают возможность применить к ним команды из заранее заданного списка. На уровне пользовательского интерфейса доступ к параметрам смарт-тегов можно получить, выбрав в меню Tools (Сервис) команду AutoCorrect Options (Параметры автозамены).

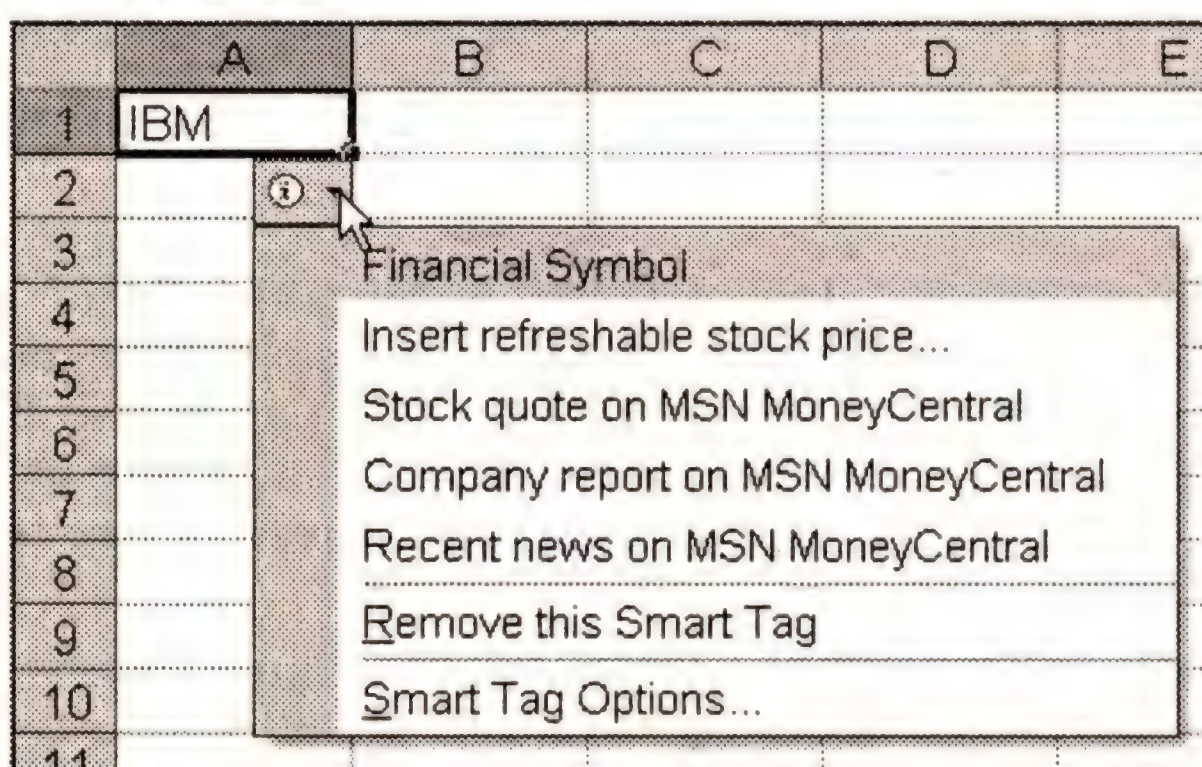


Рис. 22.2. Смарт-теги в Excel XP

Объект SmartTagRecognizer

Объект SmartTagRecognizer отвечает за распознавание данных определенного типа. Подобные устройства распознавания принимают форму кода в динамически подключаемой библиотеке (DLL — Dynamic Link Library), которая загружается в адресное пространство Excel при первом запуске на выполнение этого приложения. Например, в авторской системе следующий код:

```
Dim str As SmartTagRecognizer
For Each str In Application.SmartTagRecognizers
    Debug.Print str.progID & "/" & str.Enabled
Next
```

дает в результате:

```
FName.SmartTag/True
MOFL.SmartTag/True.
```

указывая, что в адресное пространство Excel загружены два устройства распознавания смарт-тегов. Один DLL можно использовать для распознавания нескольких типов смарт-тегов. Из кода также видно, что семейство SmartTagRecognizers содержит объект SmartTagRecognizer для каждого устройства распознавания смарт-тегов, доступного в данный момент.

Объект SmartTagRecognizer имеет редактируемое свойство Enabled, которое используется для включения и выключения устройств распознавания. Кроме того, имеются предназначенные только для чтения свойства progID и FullName, возвращающие имя устройства распознавания. Свойство FullName возвращает также полный маршрут доступа.

Объект SmartTag

Объект Range имеет свойство SmartTags, возвращающее семейство всех объектов SmartTag. Разумеется, этот объект представляет собой смарт-тег. Обсудим важные свойства и методы объекта SmartTag, перечисленные ниже. (Назначение метода Delete понятно из его названия).

- Метод Delete
- Свойство Name
- Свойство DownloadURL
- Свойство XML
- Свойство SmartTagActions

Свойство Name объекта SmartTag возвращает полное имя смарт-тега, состоящее из адреса в пространстве имен XML, за которым следует знак номера (#), а затем имя тега. Например, для смарт-тега, изображенного на рис. 22.2, код:

```
Debug.Print st.Name
```

напечатает:

```
urn:schemas-microsoft-com:office:smarttags#stockticker
```

Не путайте имя смарт-тега с его текстом. Не существует свойства или метода, которые возвращают текст смарт-тега.

Свойство DownloadURL может возвращать адрес (URL) — страницы создателя смарт-тега. По этому адресу можно скачать дополнительные команды для смарт-тегов.

Согласно документации от Microsoft, предназначенное только для чтения свойство XML объекта SmartTag «возвращает строку, представляющую фрагмент кода XML, который передается программе обработки». В любом случае для смарт-тега, изображенного на рис. 22.2, код:

```
Debug.Print st.XML
```

приведет к появлению строки:

```
<xml xmlns:fal="urn:schemas-microsoft-com:office:smarttags"><fal:stockticker>IBM</fal:stockticker></xml>.
```

которая содержит полное имя смарт-тега и его текст (IBM).

Объект SmartTagAction

Свойство SmartTagActions возвращает семейство SmartTagActions объектов SmartTagAction. Каждый такой объект представляет собой действие, выполнимое для рассматриваемого смарт-тега. Для смарт-тега, изображенного на рис. 22.2, код:

```
Dim st As SmartTag
Set st = Application.Range("A1").SmartTags(1)
For i = 1 To st.SmartTagActions.Count
    Debug.Print st.SmartTagActions(i).Name
Next
```

даст в результате:

```
Insert refreshable stock price...
LatestQuoteData
```



```
CompanyReportData
RecentNews.
```

что соответствует четырем командам с рис. 22.2. Кстати, код:

```
Dim st As SmartTag
Dim sta As SmartTagAction
For Each sta In st.SmartTagActions
    Debug.Print sta.Name
Next
```

не приводит к выводу на печать информации, хотя и не вызывает появления сообщения об ошибке!

Объект **SmartTagAction** имеет метод **Execute**, приводящий к выполнению команд. Вот код, приводящий к выполнению первой команды с рис. 22.2:

```
Sub ExecuteASmartTag()

Dim st As SmartTag
Dim sAction As String
Dim ws As Worksheet
Set ws = Application.ActiveSheet

sAction = "Insert refreshable stock price..."

' Активация смарт-тега для кода ценной бумаги.
Set st = ws.Range("A1").SmartTags(
    "urn:schemas-microsoft-com:office:smarttags#stockticker")
st.SmartTagActions(sAction).Execute

End Sub
```

Объект SmartTagOptions

Объект **SmartTagOptions** возвращается свойством **SmartTagOptions** объекта **Workbook**. Его свойство **DisplaySmartTags** может принимать значения констант из следующего перечисления:

```
Enum XlSmartTagDisplayMode
    xlIndicatorAndButton = 0
    xlDisplayNone = 1
    xlButtonOnly = 2
End Enum
```

Соответственно, можно отображать смарт-теги в виде индикатора и кнопки, только в виде кнопки или вообще никак не показывать. Логическое свойство **EmbedSmartTags** определяет, будут ли смарт-теги сохраняться вместе с книгой.

Дополнительную информацию по поводу смарт-тегов, а также о методиках их создания можно найти на веб-сайте компании Microsoft:

<http://msdn.microsoft.com/downloads/default.asp?URL=/code/sample.asp?url=/MSDN-FILES/027/001/652/msdncompositedoc.xml>

Приложения

- Приложение А. Объект Shape
- Приложение Б. Получение сведений об установленных принтерах
- Приложение В. Элементы управления командными панелями
- Приложение Г. Значки для командных кнопок
- Приложение Д. Разработка макросов Excel в среде других приложений
- Приложение Е. Языки программирования высокого и низкого уровней
- Приложение Ж. Новые объекты в Excel XP

А

Объект Shape

А теперь поговорим о том, как можно нарисовать картинку с помощью кода VBA. Так как эта тема не относится к основным вопросам программирования на языке VBA, мы не будем заострять на ней особое внимание. Впрочем, вам будет дано достаточное количество информации для дальнейшего самостоятельного изучения предмета.

Объект Shape

Каждый лист Excel и каждая диаграмма имеют специальный слой, в котором находятся рисунки. Эти рисунки представлены объектом Shape.

Логично предположить, что объекты Shape хранятся в семействе Shapes. Объекты Chart и Worksheet имеют свойство Shapes, которое возвращает семейство всех объектов Shape, нарисованных на диаграмме или листе.

Существует также объект ShapeRange, содержащий семейство выделенных объектов Shape, точно так же как объект Range может включать в себя набор выделенных ячеек. Объект ShapeRange позволяет определять свойства подсемейств всех объектов Shape.

Объект Shape с его потомками показан на рис. А.1.

Z-порядок

Каждый объект Shape имеет z-порядок, который указывает относительное положение объекта на воображаемой оси z, расположенной перпендикулярно монитору по направлению к пользователю, как показано на рис. А.2.

Предназначенное только для чтения свойство ZOrderPosition объекта Shape возвращает текущий z-порядок этого объекта. По сути, это номер объекта в семействе Shapes. Объекты Shape большего z-порядка отображаются выше объектов меньшего z-порядка. Соответственно, объект Shape, z-порядок которого равен 1, является объектом Shapes(1) и лежит в нижней части стопки!

Метод ZOrder определяет z-порядок объекта Shape относительно других объектов. Имейте в виду, что с помощью этого метода невозможно установить абсолютный z-порядок. Вот его синтаксис:

```
ShapeObject.ZOrder(ZOrderCmd).
```


где аргумент *ZOrderCmd* может принимать значения констант из следующего перечисления:

```
Enum MsoZOrderCmd
    msoBringToFront = 0
    msoSendToBack = 1
    msoBringForward = 2
    msoSendBackward = 3
End Enum
```

Это перечисление относится к объектной модели Microsoft Office.

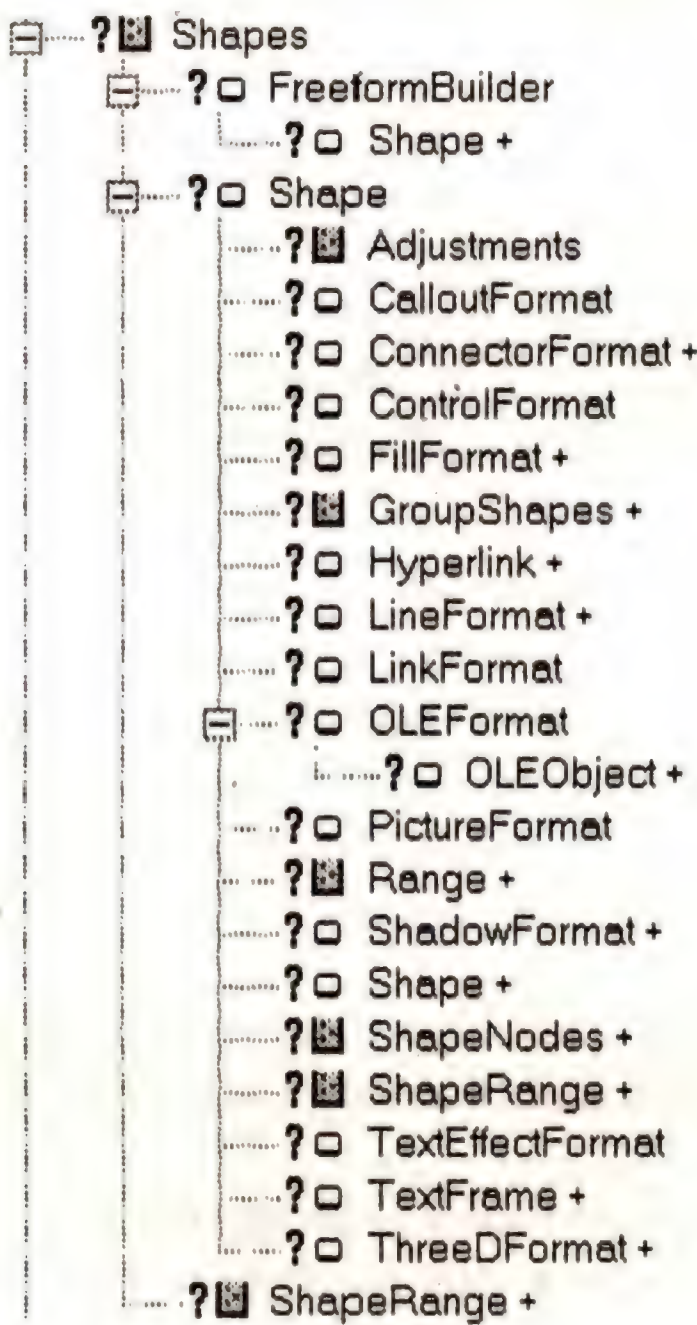


Рис. А.1. Объект Shape и его потомки

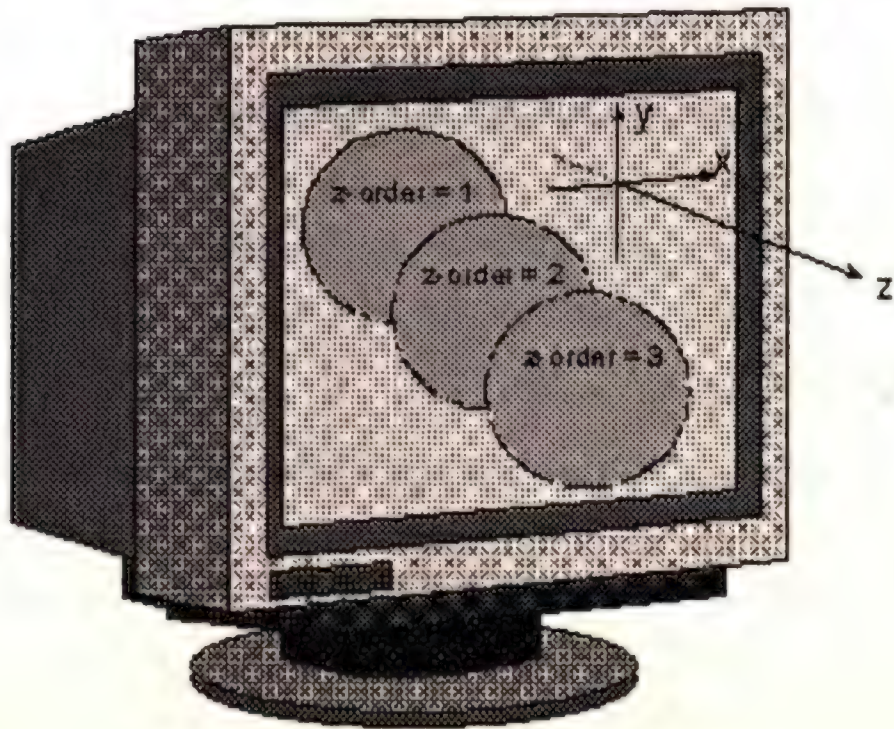


Рис. А.2. Иллюстрация z-порядка

Соответственно, согласно перечислению, процесс задания z-порядка может происходить следующим образом:

- делать z-порядок объекта максимальным;
- делать z-порядок объекта минимальным;
- увеличивать z-порядок объекта на 1;
- уменьшать z-порядок объекта на 1.

Между прочим, как вы уже успели убедиться, свойство `ZOrder` объекта `ChartObject` предназначено только для чтения. Также существуют методы `BringToFront` и `SendToBack`, меняющие z-порядок.

Создание форм

Объект `AutoShape` представляет собой объект `Shape`, включающий встроенный рисунок. Для создания новых объектов `AutoShape` используется метод `AddShape`. Вот его синтаксис:

```
ShapesObject.AddShape(Type, Left, Top, Width, Height)
```

Параметр *Type* указывает тип создаваемого объекта `AutoShape`. Он может принимать значения констант `MsoAutoShapeType`, перечисленных в табл. А.1.

Обязательные параметры *Left* и *Top* определяют положение (в точках) верхнего левого угла габаритного контейнера объекта `AutoShape`, измеренное относительно верхнего левого угла объекта-контейнера (диаграммы, листа с диаграммой или просто листа).

Параметры *Width* и *Height* указывают ширину и высоту (в точках) габаритного контейнера объекта `AutoShape`. Для изменения типа объекта `Shape` используется свойство `AutoShapeType`.

Краткая программа, текст которой содержится в листинге А.1, приводит к отображению в течение 0,5 секунды каждого объекта `AutoShape`, вместе с его свойством `AutoShapeType`. Ее следует запускать на выполнение при условии чистого активного листа. Для прерывания выполнения программы достаточно нажать комбинацию клавиш `Ctrl + Break`. Используемая при этом процедура `Delay` приведена в листинге А.2.

Листинг А.1. Отображение всех объектов AutoShape

```
Sub DisplayAutoShapes()  
  
Dim sh As Shape  
Dim i As Integer  
Set sh = ActiveSheet.Shapes.AddShape(1, 100, 100, 72, 72)  
For i = 1 To 138  
    sh.AutoShapeType = i  
    sh.Visible = True  
    ActiveSheet.Cells(1, 1).Value = sh.AutoShapeType  
    Delay 0.5  
Next i  
  
End Sub
```


Листинг А.2. Процедура Delay

```
Public Sub Delay(rTime As Single)
'Задержка rTime секунд (min=.01, max=300)
Dim OldTime As Variant
'Safty net
If rTime < 0.01 Or rTime > 300 Then rTime = 1
OldTime = Timer
Do
    DoEvents
Loop Until Timer - OldTime >= rTime
End Sub
```

Объект TextFrame

Каждый объект Shape имеет текстовую рамку, в которой содержится связанный с рисунком текст. Объект TextFrame возвращается свойством TextFrame.

Объект TextFrame имеет свойство Characters, возвращающее семейство Characters. Именно это семейство определяет текст внутри рамки. Например, код из листинга А.3 рисует на активном листе прямоугольник, добавляет в него текст и выравнивает этот текст по центру рамки.

Листинг А.3. Процедура AddRectangle

```
Sub AddRectangle()

With ActiveSheet.Shapes.AddShape(msoShapeRectangle, 10, 10, 200, 100).TextFrame
    .Characters.Text = "Это прямоугольник"
    .HorizontalAlignment = xlHAlignCenter
    .VerticalAlignment = xlVAlignCenter
End With

End Sub
```

Объект FillFormat

Объект FillFormat задает различные варианты форматирования для объекта Shape. Доступ к нему осуществляется с помощью свойства Fill объекта Shape. Среди свойств объекта FillFormat можно выделить BackColor, ForeColor, Pattern и Visible. Для определения цветовых свойств используется цветовая модель RGB:

```
sh.Fill.ForeColor.RGB = RGB(0, 0, 255)
```

Примеры

Для иллюстрации возможностей объекта AutoShapes рассмотрим листинг А.4, при выполнении которого в слой рисунков вставляется затухающая синусоида, состоящая из небольших звездочек.

Листинг А.4. Рисование затухающей синусоиды, состоящей из звездочек

```
Sub DrawSine2()
' Создание затухающей синусоиды, состоящей из звездочек
Const pi = 3.1416
Dim i As Integer
Dim x As Single, y As Single
Dim rng As Range ' Для начальной точки
```

продолжение ➤

Таблица А.1. Константы и значения MsoAutoShapeType

msoShape16pointStar (94)	msoShapeFlowchartCard (75)	msoShapeLineCallout2BorderandAccentBar (122)
msoShape24pointStar (95)	msoShapeFlowchartCollate (79)	msoShapeLineCallout2NoBorder (118)
msoShape32pointStar (96)	msoShapeFlowchartConnector (73)	msoShapeLineCallout3 (111)
msoShape4pointStar (91)	msoShapeFlowchartData (64)	msoShapeLineCallout3AccentBar (115)
msoShape5pointStar (92)	msoShapeFlowchartDecision (63)	msoShapeLineCallout3BorderandAccentBar (123)
msoShape8pointStar (93)	msoShapeFlowchartDelay (84)	msoShapeLineCallout3NoBorder (119)
msoShapeActionButtonBackorPrevious (129)	msoShapeFlowchartDirectmsoShapeLineCallout4 (112)	AccessStorage (87)
msoShapeActionButtonBeginning (131)	msoShapeFlowchartDisplay (88)	msoShapeLineCallout4AccentBar (116)
msoShapeActionButtonCustom (125)	msoShapeFlowchartDocument (67)	msoShapeLineCallout4BorderandAccentBar (124)
msoShapeActionButtonDocument (134)	msoShapeFlowchartExtract (81)	msoShapeLineCallout4NoBorder (120)
msoShapeActionButtonEnd (132)	msoShapeFlowchartInternalStorage (66)	msoShapeMixed (-2)
msoShapeActionButtonForwardorNext (130)	msoShapeFlowchartMagneticDisk (86)	msoShapeMoon (24)
msoShapeActionButtonHelp (127)	msoShapeFlowchartManualInput (71)	msoShapeNoSymbol (19)
msoShapeActionButtonHome (126)	msoShapeFlowchartManualOperation (72)	msoShapeNotchedRightArrow (50)
msoShapeActionButtonInformation (128)	msoShapeFlowchartMerge (82)	msoShapeNotPrimitive (138)
msoShapeActionButtonMovie (136)	msoShapeFlowchartMultidocument (68)	msoShapeOctagon (6)
msoShapeActionButtonReturn (133)	msoShapeFlowchartOffpageConnector (74)	msoShapeOval (9)
msoShapeActionButtonSound (135)	msoShapeFlowchartOr (78)	msoShapeOvalCallout (107)
msoShapeArc (25)	msoShapeFlowchartPredefinedProcess (65)	msoShapeParallelogram (2)
msoShapeBalloon (137)	msoShapeFlowchartPreparation (70)	msoShapePentagon (51)
msoShapeBentArrow (41)	msoShapeFlowchartProcess (61)	msoShapePlaque (28)
msoShapeBentUpArrow (44)	msoShapeFlowchartPunchedTape (76)	msoShapeQuadArrow (39)
msoShapeBevel (15)	msoShapeFlowchartSequentialAccessStorage (85)	msoShapeQuadArrowCallout (59)
msoShapeBlockArc (20)	msoShapeFlowchartSort (80)	msoShapeRectangle (1)
msoShapeCan (13)	msoShapeFlowchartStoredData (83)	msoShapeRectangularCallout (105)
msoShapeChevron (52)	msoShapeFlowchartSummingJunction (77)	msoShapeRegularPentagon (12)

Таблица А.1(продолжение)

msoShapeCircularArrow (60)	msoShapeFlowchartTerminator (69)	msoShapeRightArrow (33)
msoShapeCloudCallout (108)	msoShapeFoldedCorner (16)	msoShapeRightArrowCallout (53)
msoShapeCross (11)	msoShapeHeart (21)	msoShapeRightBrace (32)
msoShapeCube (14)	msoShapeHexagon (10)	msoShapeRightBracket (30)
msoShapeCurvedDownArrow (48)	msoShapeHorizontalScroll (102)	msoShapeRightTriangle (8)
msoShapeCurvedDownRibbon (100)	msoShapeIsoscelesTriangle (7)	msoShapeRoundedRectangle (5)
msoShapeCurvedLeftArrow (46)	msoShapeLeftArrow (34)	msoShapeRoundedRectangularCallout (106)
msoShapeCurvedRightArrow (45)	msoShapeLeftArrowCallout (54)	msoShapeSmileyFace (17)
msoShapeCurvedUpArrow (47)	msoShapeLeftBrace (31)	msoShapeStripedRightArrow (49)
msoShapeCurvedUpRibbon (99)	msoShapeLeftBracket (29)	msoShapeSun (23)
msoShapeDiamond (4)	msoShapeLeftRightArrow (37)	msoShapeTrapezoid (3)
msoShapeDonut (18)	msoShapeLeftRightArrowCallout (57)	msoShapeUpArrow (35)
msoShapeDoubleBrace (27)	msoShapeLeftRightUpArrow (40)	msoShapeUpArrowCallout (55)
msoShapeDoubleBracket (26)	msoShapeLeftUpArrow (43)	msoShapeUpDownArrow (38)
msoShapeDoubleWave (104)	msoShapeLightningBolt (22)	msoShapeUpDownArrowCallout (58)
msoShapeDownArrow (36)	msoShapeLineCallout1 (109)	msoShapeUpRibbon (97)
msoShapeDownArrowCallout (56)	msoShapeLineCallout1AccentBar (113)	msoShapeUTurnArrow (42)
msoShapeDownRibbon (98)	msoShapeLineCallout1BorderandAccentBar (121)	msoShapeVerticalScroll (101)
msoShapeExplosion1 (89)	msoShapeLineCallout1NoBorder (117)	msoShapeWave (103)
msoShapeExplosion2 (90)	msoShapeLineCallout2 (110)	
msoShapeFlowchartAlternateProcess (62)	msoShapeLineCallout2AccentBar (114)	

Листинг А.4 (продолжение)

```

Dim n As Single      ' Длина цикла в дюймах
Dim k As Integer     ' Количество звездочек равно k
Dim ScaleY As Single ' Масштабирование по вертикали
Dim sSize As Single  ' Размер звездочек
Dim sDamp1 As Single ' Фактор затухания
Dim sDamp2 As Single ' Фактор затухания
Dim cCycles As Integer ' Количество циклов
Dim sh As Shape
Dim StartLeft As Integer
Dim StartTop As Integer

' Начальное положение
StartLeft = ActiveCell.Left
StartTop = ActiveCell.Top

cCycles = 3
sDamp1 = 1
sDamp2 = 0.2
n = 2
k = 20
ScaleY = 0.5
sSize = Application.InchesToPoints(0.1)
' Зацикливание первой кривой со сдвигом по фазе
For i = 1 To cCycles * k
    x = n * i / k
    y = ScaleY * Sin((2 * pi * i) / k + n) * _
        (sDamp1 / (x + sDamp2))
    y = Application.InchesToPoints(y)
    x = Application.InchesToPoints(x)
    Set sh = ActiveSheet.Shapes.AddShape _
        (msoShape5pointStar, StartLeft + x, StartTop + y, sSize, sSize)
    sh.Fill.ForeColor.RGB = RGB(192, 192, 192) ' 25% gray
    sh.Fill.Visible = msoTrue
Next i
End Sub

```

Результат выполнения этого кода показан на рис. А.3.



Рис. А.3. Затухающая синусоида, состоящая из звездочек

Код, помещенный в листинге А.5, приводит к появлению случайных рядов звездочек. При этом внутри звездочек каждого ряда находятся буквы, складывающиеся в имя. Обратите внимание на то, что при каждом запуске программы используется новый образец.

Листинг А.5. Получение произвольных наборов звездочек

```
Sub DrawName()  
  
    ' Случайное расположение звездочек с буквами внутри  
    Const pi = 3.1416  
    Dim i As Integer  
    Dim x As Single, y As Single  
    Dim z As Single  
    Dim rng As Range      ' Для начальной точки  
    Dim n As Single       ' Длина цикла в точках  
    Dim k As Integer      ' Количество звездочек равно k  
    Dim sSize As Single   ' Размер звездочек  
    Dim sh As Shape  
    Dim sName As String   ' Отображаемое внутри имя  
    Dim StartLeft As Integer  
    Dim StartTop As Integer  
  
    ' Начальное положение  
    StartLeft = ActiveCell.Left  
    StartTop = ActiveCell.Top  
  
    sName = "Steven Roman"  
    n = 5  
    k = Len(sName)  
    sSize = Application.InchesToPoints(0.5)  
  
    Randomize Timer  
    z = 0#  
    ' Зацикливание первой кривой со сдвигом по фазе  
    For i = 1 To k  
        If Mid(sName, i, 1) <> " " Then  
            x = n * i / k  
            x = Application.InchesToPoints(x)  
  
            ' Случайное получение 0 или 1. В соответствии с этим сдвиг вверх или  
            ' вниз.  
            If Int(2 * Rnd) = 0 Then  
                z = z + 0.2  
            Else  
                z = z - 0.2  
            End If  
            y = Application.InchesToPoints(z)  
            Set sh = ActiveSheet.Shapes.AddShape _  
                (msoShape5pointStar, StartLeft + x, StartTop + y, sSize, sSize)  
  
            ' Добавление тени  
            sh.Fill.ForeColor.RGB = RGB(230, 230, 230)  
            sh.Fill.Visible = msoTrue  
  
            ' Добавление текста  
            sh.TextFrame.Characters.Text = Mid(sName, i, 1)  
            sh.TextFrame.Characters.Font.Size = 10  
            sh.TextFrame.Characters.Font.Name = "Arial"  
            sh.TextFrame.Characters.Font.Bold = True  
        End If  
    Next i  
End Sub
```


Результат выполнения этого кода показан на рис. А.4.

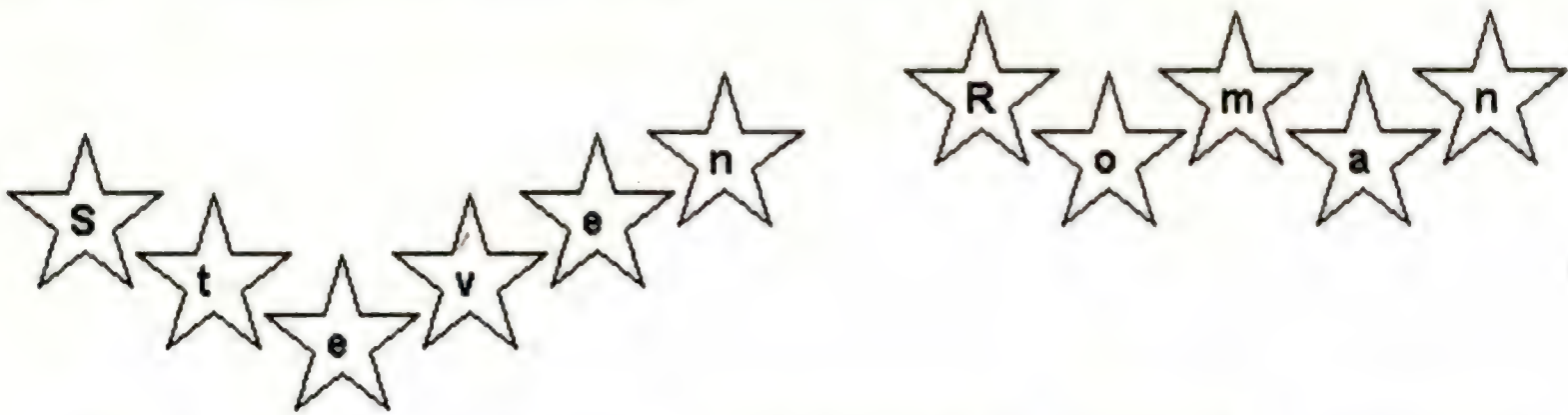


Рис. А.4. Случайным образом расположенные звездочки с буквами внутри

Код, приведенный в листинге А.6, выводит на печать гипоциклоиду. Для получения результата в данном случае может потребоваться несколько минут.

Листинг А.6. Процедура DrawHypocycloid

```
Sub DrawHypocycloid()

' Рисование гипоциклоиды, состоящей из звездочек
Const pi = 3.1416
Dim t As Single
Dim i As Integer
Dim x As Single, y As Single
Dim rng As Range ' Для начальной точки
Dim n As Single
Dim k As Integer
Dim sSize As Single ' Размер звездочек
Dim r As Integer
Dim r0 As Integer
Dim R1 As Integer
Dim sh As Shape
Dim sc As Single
Dim StartLeft As Integer
Dim StartTop As Integer

' Начальное положение
StartLeft = ActiveCell.Left
StartTop = ActiveCell.Top

r = 1
r0 = 3 * r
R1 = 8 * r
n = 400
k = 4
sc = 0.1
sSize = Application.InchesToPoints(0.03)
' Start curve at insertion point
Set rng = ActiveCell
For i = 1 To n
    t = k * pi * i / n
    x = (R1 - r) * Cos(t) + r0 * Cos(t * (R1 - r) / r)
    y = (R1 - r) * Sin(t) - r0 * Sin(t * (R1 - r) / r)
    x = sc * x
    y = sc * y
    x = Application.InchesToPoints(x)
```



```
y = Application.InchesToPoints(y)
Set sh = ActiveSheet.Shapes.AddShape
    (msoShape5pointStar, StartLeft + x, StartTop + y, sSize, sSize)
Next i
End Sub
```

Результат выполнения этого кода показан на рис. А.5. Небольшая вертикальная полоска указывает на левую границу активной ячейки

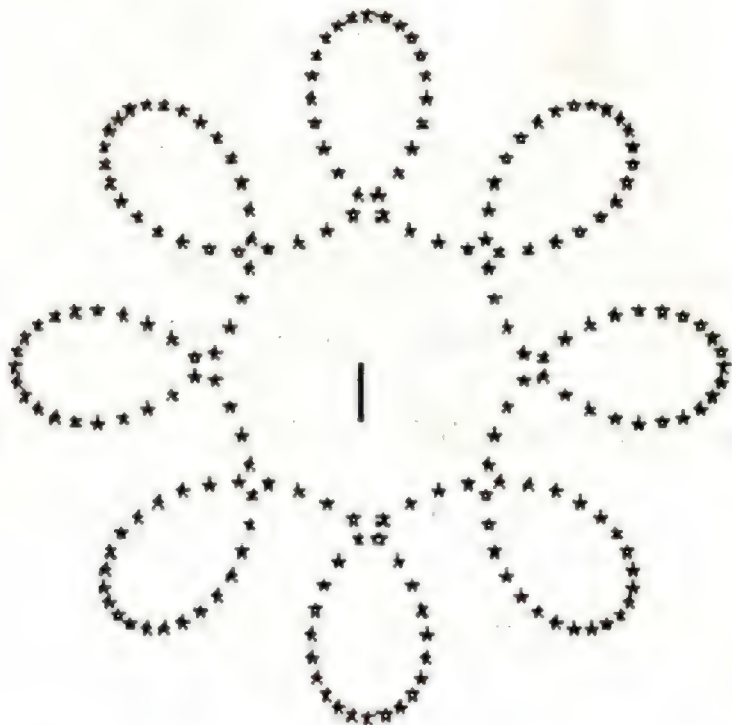


Рис. А.5. Так выглядит гипоциклоида

Объекты Diagram, DiagramNode и DiagramNodeChildren

Организационные диаграммы представляют собой специальный тип форм, предназначенный для визуальной организации данных. Пример организационной диаграммы показан на рис. А.6.

Щелкните для добавления текста

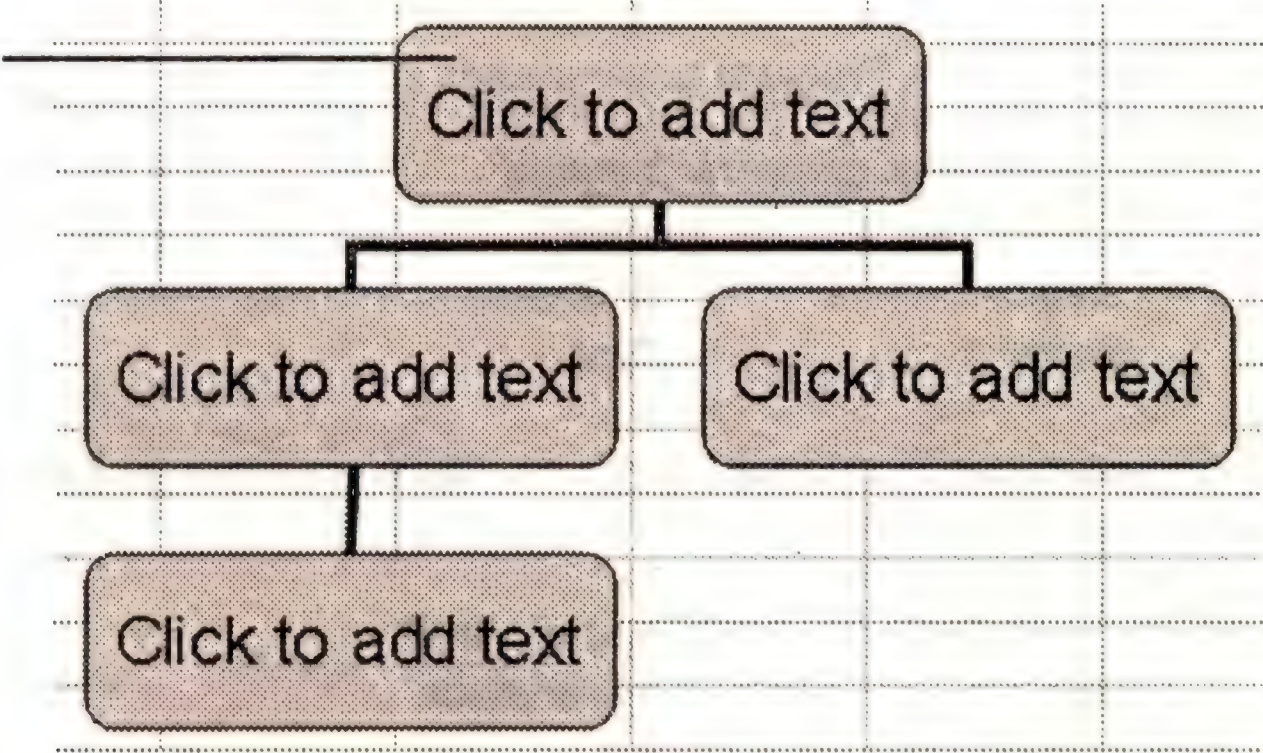


Рис. А.6. Пример организационной диаграммы

Как следует из диалогового окна Diagram Gallery (Библиотека диаграмм), показанного на рис. А.7, существует шесть типов организационных диаграмм.

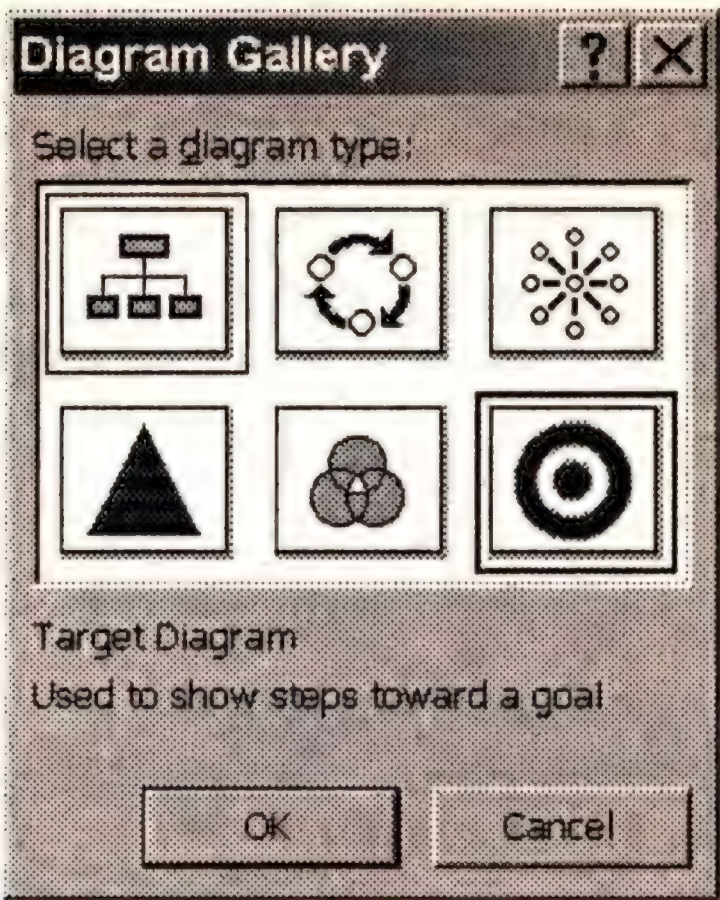


Рис. А.7. Диалоговое окно Diagram Gallery

На рис. А.8 показана часть объектной модели Excel XP, связанной с объектом Diagram.

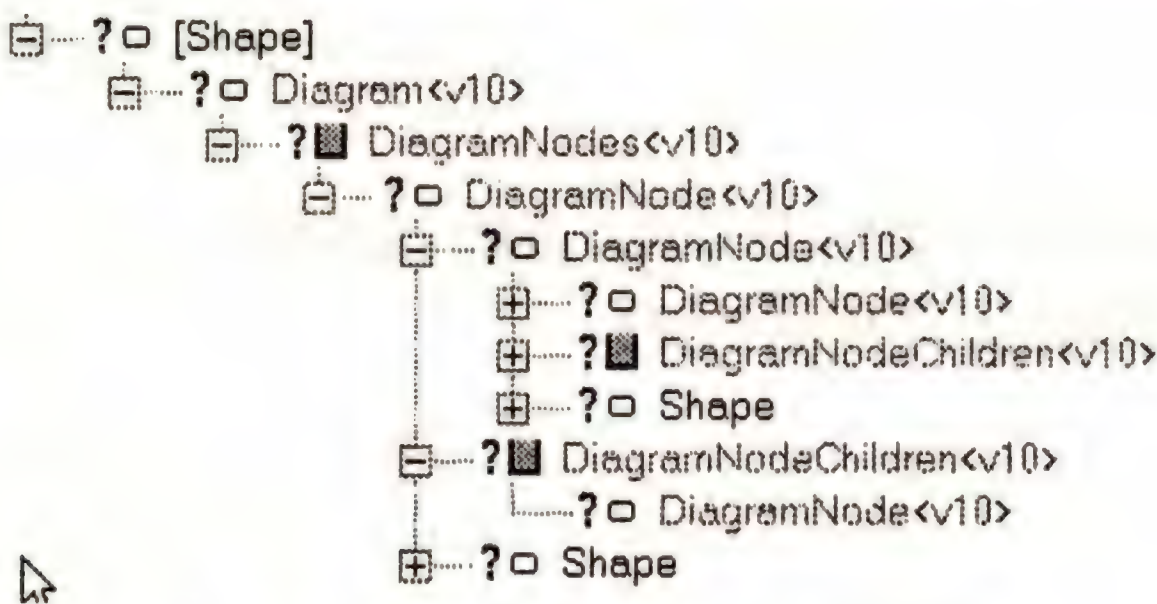


Рис. А.8. Объект Diagram и его потомки

Свойства и методы объекта Diagram перечислены в табл. А.2.

Таблица А.2. Свойства и методы объекта Diagram

Application	Convert	Parent
AutoFormat	Creator	Reverse
AutoLayout	Nodes	Type

Объект DiagramNode включает большее число свойств и методов. Они перечислены в табл. А.3.

Таблица А.3. Свойства и методы объекта DiagramNode

AddNode	Diagram	ReplaceNode
Application	Layout	Root
Children	MoveNode	Shape
CloneNode	NextNode	SwapNode
Creator	Parent	TextShape
Delete	PrevNode	TransferChildren

Разумеется, объект `DiagramNodeChildren` представляет собой семейство узлов организационной диаграммы. Свойства и методы этого объекта перечислены в табл. А.4.

Таблица А.4. Свойства и методы объекта `DiagramNodeChildren`

<code>_NewEnum</code>	<code>Creator</code>	<code>Parent</code>
<code>AddNode</code>	<code>FirstChild</code>	<code>SelectAll</code>
<code>Application</code>	<code>Item</code>	
<code>Count</code>	<code>LastChild</code>	

Для возвращения объекта `DiagramNodeChildren` можно использовать свойство `Children` объекта `DiagramNode`. Например, написав:

```
ActiveSheet.Shapes(1).Diagram.Nodes(1).Children.LastChild,
```

получим последний потомок узла.

К сожалению, при включении в код объектов, связанных с организационными диаграммами, нельзя утверждать, будут ли они работать корректно.

Чтобы создать организационную диаграмму программным способом, используется метод `AddDiagram` семейства `Shapes`. Рассмотрим код, создающий диаграмму, изображенную на рис. А.6. Имейте в виду, что добавить редактируемый текст узлам диаграммы не так-то просто. Честно говоря, у меня эта функция не работала вообще. Даже пример кода от компании Microsoft приводил к появлению сообщения об ошибке. Я даже попытался написать изменяющий текст макрос, воспользовавшись полученным таким способом кодом, но увидел только очередное сообщение об ошибке. Итак, вот этот код:

```
Sub Example_Diagram()
```

```
Dim ws As Worksheet
```

```
Set ws = Application.ActiveSheet
```

```
Dim sh As Shape
```

```
Dim dia As Diagram
```

```
Dim node1 As DiagramNode
```

```
Dim node20f1 As DiagramNode
```

```
Dim node30f1 As DiagramNode
```

```
Dim node40f2 As DiagramNode
```

```
Set sh = ws.Shapes.AddDiagram(msoDiagramOrgChart, 0, 0, 300, 300)
```

```
' Добавление первого узла. Обратите внимание на странный синтаксис, так  
' как этот узел не является потомком.
```

```
Set node1 = sh.DiagramNode.Children.AddNode
```

```
' Добавление последующих узлов
```

```
Set node20f1 = node1.Children.AddNode
```

```
Set node30f1 = node1.Children.AddNode
```

```
Set node40f2 = node20f1.Children.AddNode
```

```
' Это работает - прочитать текст можно
```

```
MsgBox node1.TextShape.TextFrame.Characters.Text
```

```
' Это не работает - отредактировать текст нельзя!!
```

```
node1.TextShape.TextFrame.Characters.Text = "Hamlet"
```

```
End Sub
```


Б

Получение сведений об установленных принтерах

Как упоминалось в главе 10, свойству `ActivePrinter` может быть присвоено имя активного принтера. Возникает вопрос определения типа принтеров, установленных на компьютере. На первый взгляд кажется, что это невозможно сделать средствами VBA. (В языке Visual Basic существует семейство `Printers`, а в языке Visual Basic for Applications подобные средства отсутствуют).

В данном приложении рассматривается программа, с помощью которой можно получить сведения относительно установленных принтеров. При этом используется набор процедур Windows API.

Первым шагом будет описание специальных констант в разделе описаний стандартного модуля:

```
Public Const KEY_ENUMERATE_SUB_KEYS = &H8
Public Const HKEY_LOCAL_MACHINE = &H80000002
Public Const SUCCESS = 0
```

Далее следует описание пользовательского типа. В данной книге не рассматривались подобные структуры данных, но, вообще говоря, пользовательским называется тип, позволяющий хранить в одной переменной несколько значений различных типов. Введите в раздел описаний следующий код:

```
Type FILETIME
    dwLowDateTime As Long
    dwHighDateTime As Long
End Type
```

Затем требуется описать три функции API. В раздел описаний введите следующий код:

```
Declare Function RegOpenKeyEx Lib "advapi32.dll" Alias _
    "RegOpenKeyExA" (ByVal hKey As Long, ByVal lpSubKey As _
    String, ByVal ulOptions As Long, ByVal samDesired As _
    Long, phkResult As Long) As Long
Declare Function RegEnumKeyEx Lib "advapi32.dll" Alias _
    "RegEnumKeyExA" (ByVal hKey As Long, ByVal dwIndex As _
    Long, ByVal lpName As String, lpcbName As Long, ByVal _
    lpReserved As Long, ByVal lpClass As String, lpcbClass _
    As Long, lpftLastWriteTime As FILETIME) As Long
Declare Function RegCloseKey Lib "advapi32.dll" _
    (ByVal hKey As Long) As Long
```


Теперь все готово к написанию основной процедуры, код которой показан в листинге Б.1. Эта процедура выбирает названия установленных принтеров из системного реестра Windows.

Листинг Б.1. Процедура GetInstalledPrinters

```
Public Sub GetInstalledPrinters(ByRef sPrinters() As _
    String, ByRef cPrinters As Long)
    ' Присваивает переменной cPrinters количество установленных принтеров.
    ' Заполняет массив sPrinters именами этих принтеров.
    Dim ft As FILETIME
    Dim KeyHandle As Long
    Dim KeyName As String
    Dim KeyLen As Long
    Dim Response As Long
    On Error GoTo ERR_INSTALLED_PRINTERS
    ReDim sPrinters(1 To 5)
    cPrinters = 0
    ' Открывает ключ системного реестра, подклкчи которого содержат
    ' сведения об установленных принтерах
    Response = RegOpenKeyEx(HKEY_LOCAL_MACHINE, _
        "SYSTEM\CurrentControlSet\Control\Print\Printers", _
        0, KEY_ENUMERATE_SUB_KEYS, KeyHandle)
    ' При наличии ошибки – сообщение о ней и выход.
    If Response <> SUCCESS Then
        MsgBox "Невозможно открыть ключ системного реестра."
        Exit Sub
    End If
    ' Цикл, в процессе которого ищутся подклкчи
    Do
        KeyLen = 1000 ' Достаточное пространство для помещения имени принтера
        KeyName = String(KeyLen, 0) ' Заполнение нулями

        Response = RegEnumKeyEx(KeyHandle, cPrinters, _
            KeyName, KeyLen, 0&, vbNullString, 0&, ft)

        ' Если операция не была выполнена, выход.
        If Response <> SUCCESS Then Exit Do

        ' Следующий свободный индекс
        cPrinters = cPrinters + 1

        ' При необходимости увеличиваем размер пространства
        If UBound(sPrinters) < cPrinters Then
            ReDim Preserve sPrinters(1 To cPrinters + 5)
        End If

        ' Добавляем в массив
        sPrinters(cPrinters) = Left(KeyName, KeyLen)
    Loop
    RegCloseKey KeyHandle
    Exit Sub
ERR_INSTALLED_PRINTERS:
    MsgBox Err.Description
    Exit Sub
End Sub
```


Процедура `GetInstalledPrinters` имеет два параметра: строковый массив `sPrinters` и переменную `cPrinters` типа `Long`. Процедура присваивает переменной `cPrinters` значение, соответствующее количеству установленных принтеров, определяя число элементов массива `sPrinters`, который затем заполняется именами принтеров.

Листинг Б.2 демонстрирует, каким образом следует использовать подпрограмму `GetInstalledPrinters`. Она присваивает имя принтера строковой переменной и отображает ее значение.

Листинг Б.2. Вызов процедуры `GetInstalledPrinters`

```
Sub DisplayInstalledPrinters()  
Dim sPrinters() As String  
Dim cPrinters As Long  
Dim i As Integer  
Dim msg As String  
' Получение информации об установленных принтерах  
GetInstalledPrinters sPrinters(), cPrinters  
' Создание сообщения и его отображение  
msg = ""  
For i = 1 To cPrinters  
    msg = msg & sPrinters(i) & vbCrLf  
Next i  
MsgBox msg, . cPrinters & " Printers"  
End Sub
```

Результат выполнения этого макроса в моей системе показан на рис. Б.1.

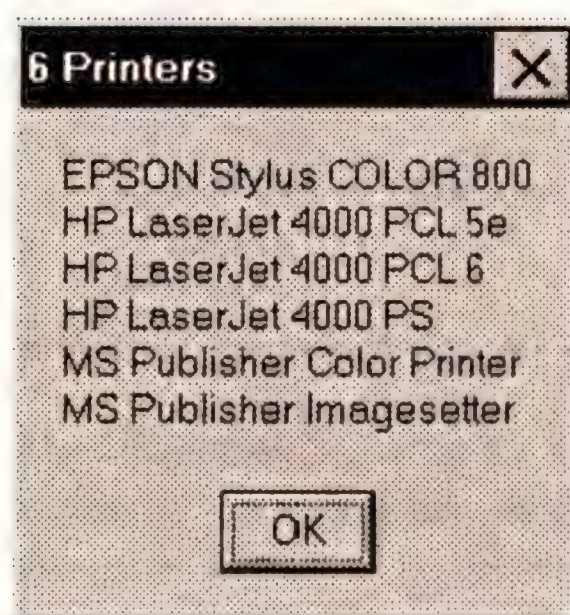


Рис. Б.1. Установленные принтеры

Предостережение: перед первым выполнением подпрограммы `GetInstalledPrinters` (посредством макроса `DisplayInstalledPrinters` или любым другим способом) сохраните файлы во всех открытых приложениях. Если в процессе ввода кода была допущена опечатка, это может привести к сбою работы всей системы и, как следствие, к потере всех несохраненных данных!

В

Элементы управления командными панелями

В этом приложении находится список всех встроенных элементов управления командными панелями (с указанием соответствующих номеров). Эти сведения требуются при работе с методом Add объекта CommandBarControls.

Встроенные элементы управления командными панелями

Таблица В.1. Элементы управления командными панелями
и соответствующие им номера

<Custom>	1	AutoFormat	107
&Spelling	2	&Format Painter	108
&Save	3	Print Pre&view	109
&Print	4	Custom	112
&New	18	&Bold	113
&Copy	19	&Italic	114
Cu&t	21	&Underline	115
&Paste	22	&Custom	117
Open	23	&Dark Shading	118
Can't Repeat	37	&Align Left	120
&Microsoft Word	42	&Align Right	121
Clear Contents	47	&Center	122
Custom	51	&Justify	123
&Piggy Bank	52	What's &This?	124
Custom	59	Custom	126
&Double Underline	60	&Undo	128
Custom	67	&Redo	129
Custom	68	&Line	130
&Close	106	Te&xt Box	139

продолжение ➤

Таблица В.1 (продолжение)

&Find File	140	Microsoft &PowerPoint	267
&Top Border	145	Microsoft P&roject	269
&Bottom Border	146	Custom	270
&Left Border	147	Custom	271
&Right Border	148	Custom	272
Apply Inside Borders	149	Custom	273
&Outside Borders	150	Custom	274
Clear Border	151	Custom	275
&Group	164	Custom	276
&Ungroup	165	Custom	278
Bring to Fron&t	166	Custom	279
Send to Bac&k	167	&Camera	280
Bring &Forward	170	Custom	281
Send &Backward	171	&Button	282
F&ull Screen	178	Custom	283
&Select Objects	182	Custom	286
&Record New Macro	184	&Strikethrough	290
&Macros	186	Delete	292
Step Macro	187	Delete &Rows	293
Resume Macro	189	Delete &Columns	294
Flip &Horizontal	196	C&ells	295
Flip &Vertical	197	&Rows	296
Rotate &Right	198	&Columns	297
Rotate &Left	199	&Arrange	298
&Freeform	200	&Split	302
&Borders	203	&New Window	303
&Drawing	204	&Accept or Reject Changes	305
&Edit Points	206	R&eplace	313
Sort &Ascending	210	Regr&oup	338
Sort Des&cending	211	&3-D	339
&Edit Box	219	Se&nd	363
&Check Box	220	&Set Print Area	364
&Combo Box	221	&Insert MS Excel 4.0 Macro	365
Control Properties	222	&Clear Formatting	368
Lock Cell	225	&Paste Formatting	369
&AutoSum	226	&Paste Values	370
&Arrow	243	&Right	371
Page Set&up	247	&Down	372
&Style	254	&Equal Sign	373
Routing Slip	259	&Plus Sign	374
Microsoft &Mail	262	&Minus Sign	375
Microsoft &Access	264	&Multiplication Sign	376
Microsoft &Schedule+	265	&Division Sign	377
Microsoft Visual &FoxPro	266	&Exponentiation Sign	378

&Left Parenthesis	379	&Default Chart	435
&Right Parenthesis	380	&Chart Wizard	436
&Colon	381	&Value Axis Gridlines	437
&Comma	382	&Category Axis Gridlines	438
&Percent Sign	383	&Legend	439
&Dollar Sign	384	&Show Outline Symbols	440
Paste Function	385	&Select Visible Cells	441
&Constrain Numeric	387	&Select Current Region	442
&XL Left Border	389	&Freeze Panes	443
&XL Right Border	390	&Zoom In	444
&XL Top Border	391	&Zoom Out	445
&Light Shading	393	&Option Button	446
&Shadow	394	&Scroll Bar	447
&Currency Style	395	&List Box	448
&Percent Style	396	&Doughnut Chart	449
&Comma Style	397	&Remove Dependent Arrows	450
&Increase Decimal	398	Trace &Dependents	451
&Decrease Decimal	399	&Remove Precedent Arrows	452
&Font Color	401	Remove &All Arrows	453
&Merge and Center	402	&Attach Note	454
&Increase Font Size	403	&Update File	455
&Decrease Font Size	404	&Toggle Read Only	456
&Vertical Text	405	&Wizard	457
&Rotate Text Up	406	Auto&Filter	458
&Rotate Text Down	407	&Refresh Data	459
Distribute &Horizontally	408	&Field	460
&Scribble	409	Show &Pages	461
Custom	417	&Show Detail	462
&Area Chart	418	Trace &Error	463
&Bar Chart	419	&Hide Detail	464
&Column Chart	420	Distribute &Vertically	465
&Stacked Column Chart	421	&Group Box	467
&Line Chart	422	&Spinner	468
&Pie Chart	423	Ta&b Order	469
&3-D Area Chart	424	&Run Dialog	470
&3-D Bar Chart	425	&Combination List-Edit	471
&3-D Clustered Column Chart	426	&Combination Drop-Down Edit	475
&3-D Column Chart	427	&Label	476
&3-D Line Chart	428	Custom	477
&3-D Pie Chart	429	Custom	478
&(XY) Scatter Chart	430	Custom	479
&3-D Surface Chart	431	Custom	480
&Radar Chart	432	Custom	481
&Volume/High-Low-Close Chart	434	Custom	482

продолжение ↗

Таблица В.1 (продолжение)

Custom	483	Assign &Macro	825
Custom	484	&Window Name Goes Here	830
Toggle Grid	485	&Recent File Name Goes Here	831
&Trace Precedents	486	&Create Microsoft Outlook Task	837
&Code	488	Mi&nimize	838
&Dialog	489	&Restore	839
Page &Break	509	&Close	840
&Options	522	&Move	841
Para&meters	537	&Size	842
H&eight	541	Ma&ximize	843
&Width	542	Save &Workspace	846
&Object	546	De&lete Sheet	847
Control T&oolbox	548	&Move or Copy Sheet	848
To &Grid	549	&Formula Bar	849
C&ancel	569	&Status Bar	850
&More Controls	642	&Worksheet	852
Align &Left	664	C&ells	855
Align &Right	665	&Goal Seek	856
Align &Top	666	Sc&enarios	857
Align &Bottom	667	Assig&n Macro	859
Align &Center	668	F&orm	860
Align &Middle	669	Su&btotals	861
&Clip Art	682	&Table	862
Free Ro&tate	688	Co&nsolidate	863
&Line Style	692	&Hide	865
&Dash Style	693	&Unhide	866
&Arrow Style	694	&Up	867
&Organization Chart	702	&Left	868
&Normal	723	&Across Worksheets	869
&Page Break Preview	724	&Series	870
&Crop	732	&Justify	871
Save &As	748	&Formats	872
Propert&ies	750	&Contents	873
E&xit	752	Co&mments	874
Paste &Special	755	&Define	878
&Go To	757	&Paste	879
Lin&ks	759	&Create	880
&Header and Footer	762	&Apply	881
&AutoFormat	786	&AutoFit	882
&AutoCorrect	793	&Hide	883
&Customize	797	&Unhide	884
&Merge Cells	798	&AutoFit Selection	885
Unmerge Cells	800	&Hide	886
T&ext to Columns	806	&Unhide	887

&Standard Width	888	&Enable Selection	991
&Rename	889	&List Formulas	992
&Hide	890	&Data	993
&Unhide	891	Label &and Data	994
&Show Auditing Toolbar	892	&Label	995
&Protect Sheet	893	&Fill Effects	1006
Protect &Workbook	894	&Angle Text Upward	1013
Relative Reference	896	&Angle Text Downward	1014
Auto&Filter	899	&Open	1015
&Show All	900	&Start Page	1016
&Advanced Filter	901	&Back	1017
&Auto Outline	904	&Forward	1018
&Clear Outline	905	&Stop Current Jump	1019
S&ettings	906	&Refresh Current Page	1020
&Series	907	Open &Favorites	1021
Sized with &Window	908	&Add to Favorites	1022
Add T&rendline	914	&Show Only Web Toolbar	1023
&Selected Object	917	&WordArt	1031
Chart &Type	918	Rerou&te Connectors	1033
3-D &View	919	&Up	1035
&Zoom	925	&Down	1036
&About Microsoft Excel	927	&Left	1037
&Sort	928	&Right	1038
&Label	932	&Curve	1041
&Lotus 1-2-3 Help	936	St&raight Connector	1042
&Answer Wizard	937	&Elbow Connector	1043
&Exchange Folder	938	C&urved Connector	1044
Add-&Ins	943	&Callouts	1047
&Insert	945	&Flowchart	1048
&Select All Sheets	946	Block &Arrows	1049
Custom &Views	950	&Stars and Banners	1050
&Background	952	&More Fill Colors	1051
&Source Data	954	&More Line Colors	1052
&Location	955	&Patterned Lines	1053
Chart &Options	956	&More Lines	1054
&Sheet List	957	&More Arrows	1055
&Calculate Now	960	&Shadow Settings	1056
&Object	961	&3-D Settings	1057
C&ells	962	&WordArt Shape	1058
&Contents and Index	983	&WordArt Alignment	1059
Microsoft Excel &Help	984	&WordArt Character Spacing	1060
&Data Table	987	&WordArt Vertical Text	1061
Series in &Columns	988	&WordArt Same Letter Heights	1063
Series in &Rows	989	&More Contrast	1064

продолжение ⇨

Таблица В.1 (продолжение)

&Less Contrast	1065	&Left Arrow	1143
&More Brightness	1066	&Up Arrow	1144
&Less Brightness	1067	&Down Arrow	1145
&Nudge Shadow Up	1068	&Left-Right Arrow	1146
&Nudge Shadow Down	1069	&Up-Down Arrow	1147
&Nudge Shadow Left	1070	&Quad Arrow	1148
&Nudge Shadow Right	1071	&Left-Right-Up Arrow	1149
&More Shadow Colors	1072	&Left-Up Arrow	1150
&More 3-D Colors	1073	&Bent-Up Arrow	1151
&Left Align	1108	&Bent Arrow	1152
&Center	1109	&U-Turn Arrow	1153
&Line Callout 4 (Border and Accent Bar)	1110	&Striped Right Arrow	1154
&Rectangle	1111	&Notched Right Arrow	1155
&Parallelogram	1112	&Pentagon	1156
&Trapezoid	1113	&Chevron	1157
&Diamond	1114	&Circular Arrow	1158
&Rounded Rectangle	1115	&Curved Right Arrow	1160
&Octagon	1116	&Curved Left Arrow	1161
&Isosceles Triangle	1117	&Curved Up Arrow	1162
&Right Triangle	1118	&Curved Down Arrow	1163
&Oval	1119	&Right Arrow Callout	1164
&Hexagon	1120	&Left Arrow Callout	1165
&Cross	1121	&Up Arrow Callout	1166
&Cube	1122	&Down Arrow Callout	1167
&Plaque	1123	&Left-Right Arrow Callout	1168
&Regular Pentagon	1124	&Up-Down Arrow Callout	1169
&Wave	1125	&Quad Arrow Callout	1170
&Can	1126	&Rectangular Callout	1172
&Vertical Scroll	1127	&Rounded Rectangular Callout	1173
&Horizontal Scroll	1128	&Oval Callout	1174
&Folded Corner	1129	&Cloud Callout	1175
&Bevel	1130	&Line Callout 2	1176
&Smiley Face	1131	&Line Callout 3	1177
&Donut	1132	&Line Callout 4	1178
&"No" Symbol	1133	&Down Ribbon	1179
&Block Arc	1134	&Up Ribbon	1180
&Left Bracket	1135	&Curved Down Ribbon	1181
&Right Bracket	1136	&Curved Up Ribbon	1182
&Left Brace	1137	&5-Point Star	1183
&Right Brace	1138	&8-Point Star	1184
&Arc	1139	&16-Point Star	1185
&Lightning Bolt	1140	&24-Point Star	1186
&Heart	1141	&32-Point Star	1187
&Right Arrow	1142	&Explosion 1	1188

&Explosion 2	1189	&Stretch Justify	1355
Flowchart: Process	1190	Very T&ight	1356
&Flowchart: Alternate Process	1191	&Tight	1357
&Flowchart: Decision	1192	&Normal	1358
&Flowchart: Data	1193	&Loose	1359
&Flowchart: Predefined Process	1194	&Very Loose	1360
&Flowchart: Internal Storage	1195	&Kern Character Pairs	1361
&Flowchart: Document	1196	&Reset Picture	1362
&Flowchart: Multidocument	1197	&Automatic	1365
&Flowchart: Terminator	1198	&Grayscale	1366
&Flowchart: Preparation	1199	&Black && White	1367
&Flowchart: Manual Input	1200	&Watermark	1368
&Flowchart: Manual Operation	1201	&3-D On/Off	1374
&Flowchart: Connector	1202	&Tilt Down	1375
&Flowchart: Off-page Connector	1203	&Tilt Up	1376
&Flowchart: Card	1204	&Tilt Left	1377
&Flowchart: Punched Tape	1205	&Tilt Right	1378
&Flowchart: Summing Junction	1206	&Depth	1379
&Flowchart: Or	1207	&Direction	1380
&Flowchart: Collate	1208	&Lighting	1382
&Flowchart: Sort	1209	&Surface	1383
&Flowchart: Extract	1210	&0 pt.	1384
&Flowchart: Merge	1211	&36 pt.	1385
&Flowchart: Stored Data	1213	&72 pt.	1386
&Flowchart: Sequential Access Storage	1214	&144 pt.	1387
&Flowchart: Magnetic Disk	1215	&288 pt.	1388
&Flowchart: Direct Access Storage	1216	&Infinity	1389
&Flowchart: Display	1217	&Perspective	1390
&Flowchart: Delay	1218	P&arallel	1391
&Line Callout 1	1219	&Bright	1392
&Line Callout 1 (Accent Bar)	1220	&Normal	1393
&Line Callout 2 (Accent Bar)	1221	&Dim	1394
&Line Callout 3 (Accent Bar)	1222	&Matte	1395
&Line Callout 4 (Accent Bar)	1223	&Plastic	1396
&Line Callout 1 (No Border)	1224	Me&tal	1398
&Line Callout 2 (No Border)	1225	&Wire Frame	1399
&Line Callout 3 (No Border)	1226	Edit Te&xt	1401
&Line Callout 4 (No Border)	1227	To &Shape	1402
&Line Callout 1 (Border and Accent Bar)	1228	&Image Control	1403
&Line Callout 2 (Border and Accent Bar)	1229	&Basic Shapes	1405
&Line Callout 3 (Border and Accent Bar)	1230	&Callouts	1406
&Right Align	1352	&Flowchart	1407
Le&tter Justify	1353	Block &Arrows	1408
&Word Justify	1354	&Stars and Banners	1409

продолжение ⇨

Таблица В.1 (продолжение)

Show WordArt Toolba&r	1410	&Options	1604
&Exit Rotate Mode	1412	&Design Mode	1605
Show Picture Toolba&r	1413	&WordArt Gallery	1606
&Add Point	1415	&Chart Type	1616
De&lete Point	1416	&Pattern	1617
&Straight Segment	1417	&Border	1618
&Curved Segment	1418	&Chart Objects:	1622
Close C&urve	1420	&Navigate Circular Reference:	1623
Au&to Point	1421	&Bubble Chart	1635
&Smooth Point	1422	&3-D Cylinder Chart	1636
St&raight Point	1423	&3-D Pyramid Chart	1637
&Corner Point	1424	&3-D Cone Chart	1638
Exit &Edit Point	1425	&Double Arrow	1639
&No Color	1453	&Fill Color	1691
&Color Scheme	1455	&Line Color	1692
&Standard Colors	1456	&Shadow Color	1693
&Automatic Color	1459	&3-D Color	1694
&Color Scheme	1460	&Visual Basic Editor	1695
&Standard Colors	1461	&Bottom Double Border	1699
&Color Scheme	1465	&Font:	1728
&Standard Colors	1466	&Font Size:	1731
&Color Scheme	1470	&Style:	1732
&Standard Colors	1471	&Zoom:	1733
&View Code	1561	&Scenario:	1737
Open in New &Window	1574	&Address:	1740
&Copy Hyperlink	1575	M&ap	1741
Hyperl&ink	1576	&Merge Across	1742
Edit &Hyperlink	1577	>From &Scanner	1764
&Select Hyperlink	1578	&Inside Horizontal Border	1840
&Add to Print Area	1583	&Inside Vertical Border	1841
&Clear Print Area	1584	&Diagonal Down Border	1842
Reset All Page &Breaks	1585	&Diagonal Up Border	1843
&Exclude From Print Area	1586	&Check Box	1848
Insert Page &Break	1588	&Find	1849
Co&mment	1589	&Text Box	1850
&Previous Comment	1590	&Command Button	1851
&Next Comment	1591	&Option Button	1852
Delete Co&mment	1592	&List Box	1853
Sh&ow Comment	1593	&Combo Box	1854
Hide All Comments	1594	&Toggle Button	1855
Calculated &Field	1597	&Spin Button	1856
Calculated &Item	1598	&Scroll Bar	1857
Entire &Table	1600	&Label	1858
[[TBTIP_SxUpdateM]]	1601	&Semitransparent Shadow	1859

Search the &Web	1922	Merge &Workbooks	2044
&Automatic Color	1926	&Copy Here	2045
&Color Scheme	1927	&Move Here	2046
&Color	1928	Copy Here as &Formats Only	2047
&Color Scheme	1929	Copy Here as &Values Only	2048
&Color	1930	&Shift Down and Copy	2049
Run &Web Query	1948	Shif&t Right and Copy	2050
Run &Database Query	1949	Shift &Down and Move	2051
&Edit Query	1950	Shift &Right and Move	2052
D&ata Range Properties	1951	Create &New Query	2054
Refresh &All	1952	&Clear Validation Circles	2055
&Cancel Refresh	1953	&Image	2089
&Refresh Status	1954	&Straight Connector	2091
C&hart	1957	&Elbow Connector	2092
&Add Data	1963	C&urved Connector	2093
&All	1964	Edit Te&xt	2094
&<verb>	1965	&Link Here	2158
Pic&k From List	1966	Create &Hyperlink Here	2159
Con&vert	1967	&Shadow On/Off	2175
&Ungroup Sheets	1968	Set AutoShape &Defaults	2179
&Pattern	1988	Stop &Recording	2186
&None	2012	&Mail Recipient	2188
&Average	2013	&New	2520
&Count	2014	Print (HP LaserJet 4000 Series PS)	2521
C&ount Nums	2015	Ch&art Window	2571
&Max	2016	&From File	2619
M&in	2017	&AutoShapes	2630
&Sum	2018	&Lines	2631
&Copy Cells	2019	&Basic Shapes	2632
Fill &Series	2020	Co&nnectors	2633
Fill &Formats	2021	&Sun	2634
Fill &Values	2022	&Moon	2635
Fill &Days	2023	&Double Bracket	2636
Fill &Weekdays	2024	&Double Brace	2637
Fill &Months	2025	&4-Point Star	2638
Fill &Years	2026	&Double Wave	2639
&Linear Trend	2027	&Straight Arrow Connector	2640
&Growth Trend	2028	&Straight Double-Arrow Connector	2641
S&eries	2029	&Elbow Arrow Connector	2642
Insert Co&mment	2031	&Elbow Double-Arrow Connector	2643
Va&lidity	2034	&Curved Arrow Connector	2644
Circle Invalid Data	2035	&Curved Double-Arrow Connector	2645
R&eset Print Area	2038	&Fill Effects	2668
S&hare Workbook	2040	De&lete Segment	2692
&Highlight Changes	2042	Open C&urve	2693

продолжение ↗

Таблица В.1 (продолжение)

Paste as &Hyperlink	2787	Send Feedbac&k	3030
&Set Transparent Color	2827	&Best of the Web	3031
&Solve Order	2914	Search the &Web	3032
&PivotTable Report	2915	Web &Tutorial	3033
&Web Toolbar	2934	&Web Help 14	3034
&Custom Menu Item	2949	&Web Help 15	3035
&Custom Button	2950	&Web Help 16	3036
&Free Stuff	3021	Con&ditional Formatting	3058
&Product News	3022	Protect and &Share Workbook	3059
Frequently Asked &Questions	3023	Microsoft &Home Page	3091
Online &Support	3024	Clear Co&ntents	3125
&Web Help 5	3025	&Automatic Color	3141
&Web Help 6	3026	&Group	3159
&Web Help 7	3027	&Ungroup	3160
&Web Help 8	3028	&Increase Indent	3161
Microsoft &Office Home Page	3029	&Decrease Indent	3162

Г

Значки для командных кнопок

Свойство `FaceID` объекта `CommandBarButton` определяет значки, соответствующие командным кнопкам. Пример использования этого свойства можно найти в разделе «Пример: создание панели инструментов» главы 12.

На рис. Г.1–Г.5 показаны значки, доступные в редакторе Visual Basic, вместе с их номерами. На каждом рисунке изображены 400 значков, начальный и конечный номера которых показаны в подписи. Кроме того, чтобы облегчить нахождение определенного номера, на рисунки была наложена пронумерованная сетка. Номера столбцов указывают первую цифру, номера строк — все остальные значимые цифры. Например, свойство `faceID` значка в виде буквы F на рис. Г.1 имеет значение 85, так как этот значок находится в строке 8х (строки, содержащие значения параметра `faceID`, равного 80–89) и в столбце х5 (столбец, содержащий значения параметра `faceID`, первая цифра которого равна 5).

Обратите внимание, что некоторым числам не соответствуют какие-либо значки.

	0x	1x	2x	3x	4x	5x	6x	7x	8x	9x
0x										
1x										
2x										
3x										
4x										
5x										
6x										
7x	0	1	2	3	4	5	6	7	8	9
8x	A	B	C	D	E	F	G	H	I	J
9x	K	L	M	N	O	P	Q	R	S	T
10x	U	V	W	X	Y	Z				
11x										
12x										
13x										
14x										
15x										
16x										
17x										
18x										
19x										
20x										
21x										abl
22x										
23x										
24x										
25x										
26x	AB									
27x										
28x										
29x	ABC	ABC								
30x										
31x										
32x										
33x										
34x										
35x										
36x										
37x				"=	"+	"-	"*	"/"	"^	"("
38x	"")	":	","	"%	"\$	"f	"X	"p	"H	"I
39x						\$	%	,	∞	∞

	0x	1x	2x	3x	4x	5x	6x	7x	8x	9x
40x										
41x										
42x										
43x										
44x										
45x										
46x										
47x										
48x										
49x										
50x										
51x										
52x										
53x										
54x										
55x										
56x										
57x										
58x										
59x										
60x										
61x										
62x										
63x										
64x										
65x										
66x										
67x										
68x										
69x										
70x										
71x										
72x										
73x										
74x										
75x										
76x										
77x										
78x										

	0x	1x	2x	3x	4x	5x	6x	7x	8x	9x
80x										
81x										
82x										
83x										
84x										
85x										
86x										
87x										
88x										
89x										
90x										
91x										
92x										
93x										
94x										
95x										
96x										
97x										
98x										
99x										
100x										
101x										
102x										
103x										
104x										
105x										
106x										
107x										
108x										
109x										
100x										
111x										
112x										
113x										
114x										
115x										
116x										
117x										
118x										
119x										
120x										
121x										
122x										
123x										
124x										
125x										
126x										
127x										
128x										
129x										
130x										
131x										
132x										
133x										
134x										
135x										
136x										
137x										
138x										
139x										
140x										
141x										
142x										
143x										
144x										
145x										
146x										
147x										
148x										
149x										
150x										
151x										
152x										
153x										
154x										
155x										
156x										
157x										
158x										
159x										

Рис. Г.2. Значения свойства FaceID для значков 800–1599

	0x	1x	2x	3x	4x	5x	6x	7x	8x	9x
160x										
161x										
162x										
163x										
164x										
165x										
166x										
167x										
168x										
169x										
170x										
171x										
172x										
173x										
174x										
175x										
176x										
177x										
178x										
179x										
180x										
181x										
182x										
183x										
184x										
185x										
186x										
187x										
188x										
189x										
190x										
191x										
192x										
193x										
194x										
195x										
196x										
197x										
198x										
199x										

	0x	1x	2x	3x	4x	5x	6x	7x	8x	9x
200x										
201x										
202x										
203x										
204x										
205x										
206x										
207x										
208x										
209x										
210x										
211x										
212x										
213x										
214x										
215x										
216x										
217x										
218x										
219x										
220x										
221x										
222x										
223x										
224x										
225x										
226x										
227x										
228x										
229x										
230x										
231x										
232x										
233x										
234x										
235x										
236x										
237x										
238x										
239x										

Рис. Г.3. Значения свойства FaceID для значков 1600–2399

	0x	1x	2x	3x	4x	5x	6x	7x	8x	9x
240x										
241x										
242x										
243x										
244x										
245x										
246x										
247x										
248x										
249x										
250x										
251x										
252x										
253x										
254x										
255x										
256x										
257x										
258x										
259x										
260x										
261x										
262x										
263x										
264x										
265x										
266x										
267x										
268x										
269x										
270x										
271x										
272x										
273x										
274x										
275x										
276x										
277x										
278x										
279x										

	0x	1x	2x	3x	4x	5x	6x	7x	8x	9x
280x										
281x										
282x										
283x										
284x										
285x										
286x										
287x										
288x										
289x										
290x										
291x										
292x										
293x										
294x										
295x										
296x										
297x										
298x										
299x										
300x										
301x										
302x										
303x										
304x										
305x										
306x										
307x										
308x										
309x										
310x										
311x										
312x										
313x										
314x										
315x										
316x										
317x										
318x										
319x										

Рис. Г.4. Значения свойства FaceID для значков 2400–3199

	0x	1x	2x	3x	4x	5x	6x	7x	8x	9x
320x										
321x										
322x										
323x										
324x										
325x										
326x						X				
327x										
328x										
329x										
330x										
331x										
332x										
333x										
334x										
335x										
336x						AB ¹				
337x										
338x										
339x										

Рис. Г.5. Значения свойства FaceID для значков 3200–3399



Разработка макросов Excel в среде других приложений

В этом приложении будут кратко описаны способы работы с объектной моделью Excel из других приложений, в том числе из Microsoft Access, Word и PowerPoint.

Общеизвестная методика, которую фирма Microsoft называет *автоматизацией* (ранее она называлась OLE-автоматизацией), позволяет прикладной программе-серверу (*сервер автоматизации*) предоставлять свои услуги другим программам-клиентам (*контроллеры/клиенты автоматизации*) через специальный интерфейс. Так как приложения Microsoft Word, Access, Excel и PowerPoint могут являться как серверами, так и контроллерами автоматизации, пользователь вправе программировать любое из этих приложений взаимнообразно (одно из другого).

Ссылки на объектную модель Excel

Первым шагом в управлении объектной моделью Excel является установление ссылки на ее библиотеку. Редактор VBA любой из программ-клиентов (Word, Access, Excel и PowerPoint) имеет команду References в меню Tools. Выбор этой команды приводит к появлению диалогового окна References, показанного на рис. Д.1. Здесь можно выбрать объектную модель, к которой требуется получить доступ с помощью контроллера автоматизации.

Ссылки на объект Application в Excel

После установления нужной ссылки можно описать объектную переменную типа Application:

```
Dim XlApp As Excel.Application.
```

которая будет «понятной» клиенту автоматизации, так как он получит доступ к объектной библиотеке сервера. Обратите внимание на то, что в данном случае требуется указать полное имя объекта, так как объект Application присутствует и в других объектных моделях.

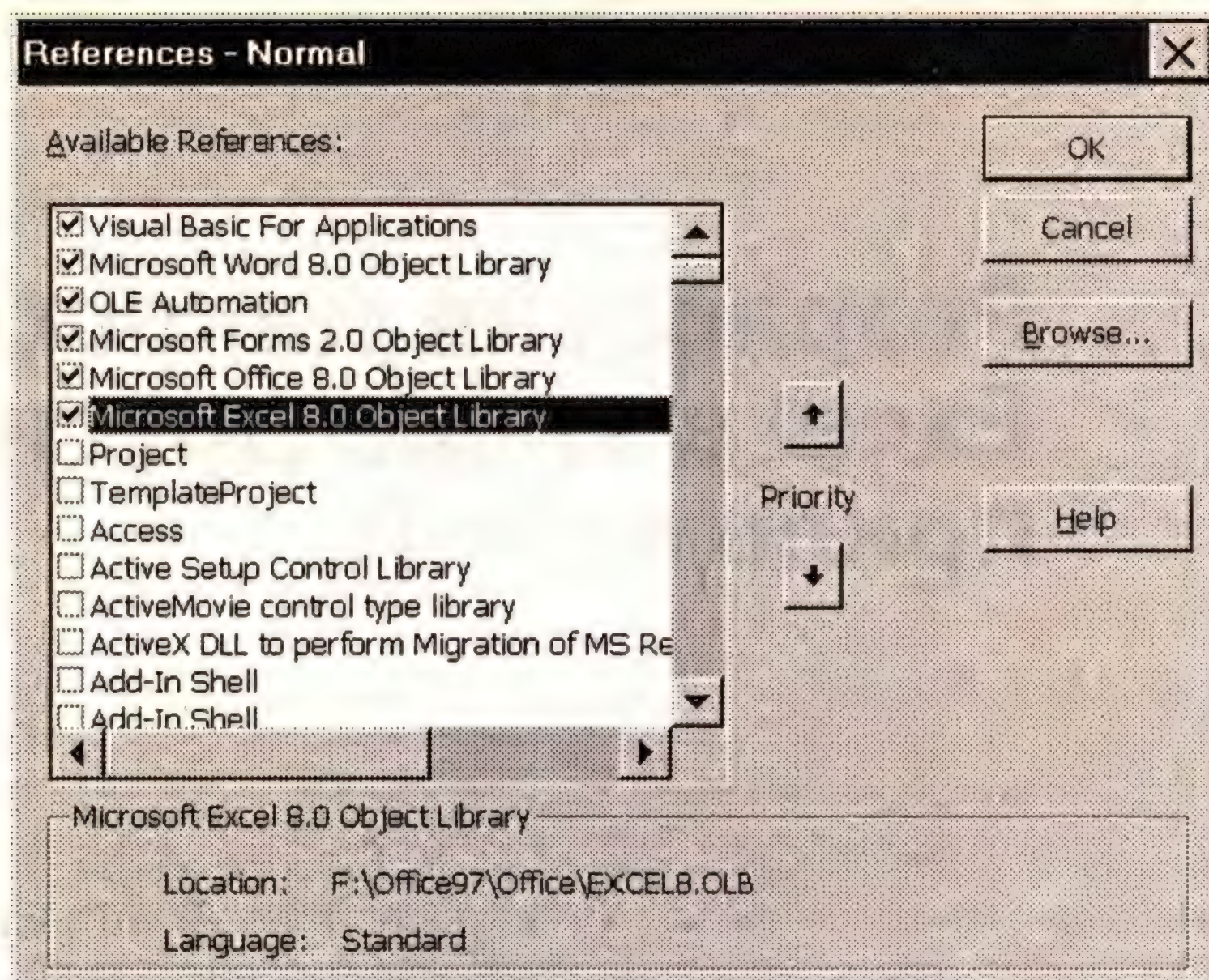


Рис. Д.1. Диалоговое окно References

Затем требуется запустить сервер автоматизации Excel, создать объект Application из объектной модели Excel и получить ссылку на этот объект. Это делается с помощью следующего кода:

```
Set XLApp = New Excel.Application
```

Начиная с этого момента пользователь получает полный доступ к объектной модели Excel. Однако важно понимать, что предыдущая строка, начиная выполнение сервера автоматизации Excel, не приводит к одновременному запуску графического интерфейса пользователя. Чтобы отобразить окно Excel, достаточно присвоить свойству Visible значение True:

```
XLApp.Visible = True
```

Теперь можно программировать, как если бы мы находились в редакторе VBA. Например, следующий код создает новую книгу, добавляет в нее лист, помещает значение в ячейку A1, а затем сохраняет книгу:

```
Sub MakeWorkbook()
```

```
Dim XLApp As Excel.Application
Dim wb As Excel.Workbook
Dim ws As Excel.Worksheet
```

```
Set XLApp = New Excel.Application
XLApp.Visible = True
```

```
Set wb = XLApp.Workbooks.Add
Set ws = wb.Worksheets.Add
ws.Name = "Sales"
ws.Range("A1").Value = 123
```



```
wb.SaveAs "d:\temp\SalesBook"
```

```
End Sub
```

Обратите внимание: сервер Excel не завершает работу сам по себе даже после уничтожения переменной XLApp. Если приложение Excel предварительно было сделано видимым, его можно закрыть как программно, так и обычным способом через пользовательский интерфейс (например, выбрав команду Exit из меню File). Скрытый сервер Excel закрывается только с помощью метода Quit:

```
XLApp.Quit
```

Если не завершить работу сервера Excel, он будет потреблять системные ресурсы компьютера до момента перезагрузки компьютера.

Альтернативный подход

Вышеописанный подход к программированию макросов Excel из других приложений является наиболее предпочтительным, так как он самый эффективный. Однако существует альтернативный вариант действий, который тоже не стоит сбрасывать со счетов. Как и раньше, предполагается, что ссылка на объектную библиотеку Excel уже установлена.

Функция CreateObject

Функция CreateObject запускает сервер автоматизации, создает объект и сопоставляет с ним объектную переменную. Соответственно, можно написать:

```
Dim XLApp as Excel.Application  
Set XLApp = CreateObject("Excel.Application")
```

В данном случае выполнение процедуры займет больше времени, но, тем не менее, такой подход тоже имеет право на существование.

Разумеется, после завершения работы приложение Excel требуется закрыть с помощью метода Quit (или, если окно Excel является видимым, через пользовательский интерфейс).

Функция GetObject

Если приложение Excel уже запущено на выполнение, функция CreateObject приведет к появлению дополнительной копии сервера Excel. Чтобы использовать приложение, запущенное в данный момент, применяется функция GetObject. Она определяет ссылку на объект Application в выполняющейся копии Excel. Это делается следующим образом:

```
Set XLApp = GetObject(, "Excel.Application")
```

(В данном случае не используется первый параметр функции GetObject).

Основная проблема применения функции GetObject состоит в том, что если приложение Excel не запущено на момент обращения к ней, появится сообщение об ошибке. Соответственно, нам требуется код, который будет запускать приложение, если оно неактивно, и использовать существующую копию Excel в противном случае.

Хитрость состоит в том, что номер ошибки, возникающей, если функция не найдет активной копии Excel, уже известен. Он равен 429 ("ActiveX component can't create object"). Соответственно, остается написать:


```
Dim XLApp As Excel.Application
On Error Resume Next
' Попытка получить ссылку на активную копию Excel
Set XLApp = GetObject(, "Excel.Application")
If Err.Number = 429 Then
    ' При ошибке номер 429 создается новый объект
    Set XLApp = CreateObject("Excel.Application")
ElseIf Err.Number <> 0 Then
    ' Сообщение об ошибке другого типа
    MsgBox "Ошибка: " & Err.Description
    Exit Sub
End If
```

Отсутствие ссылки на библиотеку объектов

Изначально предполагалось, что клиентское приложение имеет ссылку на библиотеку объектов сервера. Но для клиента автоматизации существует возможность программировать объекты сервера автоматизации и без такой ссылки. Просто при этих обстоятельствах невозможно сослаться на объект по его имени, потому что клиент просто не найдет это имя. Вместо этого используется общий объектный тип данных, как в следующем примере:

```
Dim XLApp As Object
Dim wb As Object
Set XLApp = CreateObject("Excel.Application")
XLApp.Visible = True
Set wb = XLApp.Workbooks.Add
wb.SaveAs "d:\temp\SalesBook"
```

Этот код выполняется еще медленней, чем две предыдущие версии.

Соответственно, существуют три способа автоматизации:

- с помощью ключевого слова `New` (требуется ссылка на библиотеку объектов);
- с помощью функции `CreateObject` и специального описания объектных переменных (требуется ссылка на библиотеку объектов);
- с помощью функции `CreateObject` с описанием данных общего типа как объектной переменной (не требуется ссылка на библиотеку объектов).

Иногда эти версии автоматизации называют очень ранним связыванием, ранним связыванием и динамическим связыванием соответственно. Эти термины относятся ко времени, когда редактор VBA может связать имена объектов, свойств и методов с реальными адресами названных элементов. В первом случае связывание осуществляется во время компиляции, то есть перед запуском программы. Просто раннее связывание частично осуществляется в процессе компиляции, а частично — в процессе запуска. При динамическом связывании сопоставление всех элементов с их реальными адресами происходит исключительно в процессе выполнения кода.

Ситуация очевидна. Большая степень связывания при выполнении программы приводит к замедлению ее работы. Соответственно, наиболее эффективным методом будет очень раннее связывание.

Е Языки программирования высокого и низкого уровней

В этом приложении будет произведена сравнительная оценка языка Visual Basic с «точки зрения» различных языков программирования высокого и низкого уровней.

Языки программирования низкого уровня характеризуются возможностью непосредственного управления операционной системой и аппаратным обеспечением. К примеру, они позволяют включить двигатель дисководов, проверить состояние индикатора интерфейса принтера или отдельные дисковые секторы. Все эти задачи достаточно сложно, а иногда и просто невозможно выполнить с помощью языков программирования высокого уровня. Кроме того, программа, написанная на языке программирования низкого уровня, как правило, выполняется быстрее.

С другой стороны, за возможность управлять компьютером на низком уровне приходится платить свою цену. Языки низкого уровня обычно труднее в понимании. Они намного сильнее отличаются от обычного разговорного языка и, как следствие, хуже поддаются изучению и сложнее в применении. Языки программирования высокого уровня (а также языки уровня приложения, которые многие пользователи также называют языками высокого уровня) обычно более удобны для работы, но за это приходится платить снижением степени контроля над компьютером и замедлением процесса выполнения программ.

Для иллюстрации рассмотрим процесс вывода текста на печать. Языки программирования низкого уровня могут лишь инициировать пересылку на принтер отдельных символов. Процесс печати осуществляется следующим образом:

1. Проверка состояния принтера.
2. Если принтер свободен, следует его инициализация.
3. Отправка символа на принтер.
4. Проверка получения символа.
5. Если символ не получен, он посылается снова.
6. Отправка следующего символа.

Язык наиболее низкого уровня, используемый в программировании, называется ассемблером. Он обеспечивает практически полный контроль над аппаратным обеспечением компьютера. Для иллюстрации рассмотрим код на языке ассемблера, который выводит на печать сообщение «Счастливой печати». Не волнуйтесь, если перечисленные операторы кажутся вам бессмысленным набором символов, их достаточно просто бегло просмотреть. Я всего лишь хотел продемонстрировать вам, что языки программирования низкого уровня более трудны для понимания, чем языки программирования высокого уровня. Строки, начинающиеся с точки с запятой, являются комментариями. Для экономии места в этот код не включен модуль проверки ошибок.

```
; -----
; Данные для программы
; -----
; Распечатываемое сообщение
Message DB 'Счастливой печати', 0Dh, 0Ah

; Длина сообщения
Msg_Len EQU $-Message

; -----
; Инициализация принтера 0
; -----
mov ah,1
mov dx,0
int 17h

; -----
; Печать инструкций
; -----
; Получение числа распечатываемых символов
mov cx,Msg_Len

; Получение сведений о месте расположения сообщения
mov bx,offset Message

; Получение имени принтера (первым является принтер под номером 0)
mov dx,0

Print_Loop:

; Отправка символа на принтер номер 0
mov ah,0
mov al,[bx]
int 17h

; Отправка следующего символа
inc bx
loop Print_Loop
```

Для сравнения рассмотрим, как та же самая задача будет выглядеть на языке BASIC:

```
LPRINT "Счастливой печати"
```

Как видите, разница совершенно очевидна.

Как уже упоминалось, языки программирования высокого уровня разработаны для определенных целей, как правило, для написания программных приложе-

ний. Например, языки Visual C++ и Visual Basic в основном используются для создания автономных Windows-приложений. Как известно, программа Microsoft Excel написана на языке Visual C++. С другой стороны, язык FORTRAN (сокращение словосочетания *Formula Translation* — преобразование формул) предназначен для разработки научных и вычислительных приложений на различных платформах (включая Windows). Язык COBOL предназначен для написания бизнес-приложений (для мэйнфреймов).

На высшем уровне иерархии языков программирования находятся такие языки, как Excel VBA, чьей основной целью является управление приложением высокого уровня (например, Microsoft Excel).

А теперь вкратце рассмотрим некоторые широко используемые языки программирования.

BASIC

Слово BASIC представляет собой аббревиатуру от словосочетания *Beginners All-Purpose Symbolic Instruction Code* (Код создания символических инструкций общего назначения для начинающих). Ключевым в данном случае является слово *Beginners* (начинающие). Язык BASIC был разработан в Дортмундском колледже в 1963 году двумя математиками: Джоном Кемени (John Kemeny) и Томасом Курцем (Thomas Kurtz). Их целью было создание языка программирования, предназначенного для студентов, изучающих гуманитарные науки. Соответственно, требовался интуитивно понятный пользователям язык, который можно было эффективно использовать для выполнения домашних заданий. В те дни программу приходилось передавать оператору, помещавшему ее в очередь на выполнение. Затем студенту оставалось ждать результатов. Персональных компьютеров в 60-х годах еще не было!

Первая версия языка BASIC была очень простой; скажем больше: она была примитивной. Например, в нем присутствовал только один тип данных: с плавающей точкой (различные типы данных рассматриваются в главе 5). С тех пор BASIC быстро превратился в язык программирования для персональных компьютеров, в основном благодаря поддержке компании Microsoft.

Но, однако, даже в наше время язык BASIC и его версия Visual Basic не получили широкого распространения среди специалистов в области программирования, поскольку его репутация не слишком хороша. Во-первых, он обеспечивает недостаточный уровень контроля над аппаратным обеспечением (или операционной системой), по меньшей мере, по сравнению с такими языками, как C. Во-вторых, BASIC допускает различные «вольности», что несовместимо с хорошим стилем программирования.

В целях сравнения с другими языками программирования рассмотрим программу на BASIC, которая выдает запрос пользователю о вводе чисел, а затем вычисляет их среднее арифметическое. Строки, начинающиеся с апострофа, являются комментариями.

```
' Программа на языке BASIC для вычисления  
' среднего из максимум 100 чисел
```

```
' Количество вводимых чисел
```



```
INPUT "Введите количество чисел: ", Num

' Если параметр Num больше 1, но меньше 100, продолжить
IF Num > 0 AND Num <= 100 THEN

    Sum = 0
    ' Цикл, накапливающий числа для усреднения
    FOR I = 1 TO Num
        ' Ввод следующего числа
        INPUT "Введите следующее число: ", NextNum
        ' Добавление числа к уже имеющейся сумме
        Sum = Sum + NextNum
    NEXT I

    ' Вычисление среднего арифметического
    Ave = Sum / Num

    ' Отображение среднего арифметического
    PRINT "Среднее арифметическое: "; Ave

END IF

END
```

Visual Basic

Благодаря появлению языка Visual Basic, разработанного фирмой Microsoft, позиции BASIC значительно упрочились. В начале 90-х годов перед компанией Microsoft возникла дилемма. Операционная система Windows была (и остается) значительно сложнее операционной системы DOS, поэтому только профессиональные программисты могли эффективно использовать имеющийся инструмент, основанный на языке C, для создания Windows-приложений. В наши дни для этого применяется такой инструмент, как библиотека фундаментальных классов Microsoft (MFC), который тоже вряд ли подходит для рядового программиста.

Однако экономические соображения приводили к неизбежному выводу о том, что возможность разработки Windows-приложений следует предоставить всем категориям пользователей. Поэтому в 1991 году появился язык Visual Basic (сокращенно VB), сочетающий возможности языка BASIC и визуальной среды программирования. Благодаря этому пользователи получили возможность создания графических компонентов: окна, командные кнопки, текстовые поля, переключатели, меню и прочие элементы Windows-приложений. Лежащий в основе этих компонентов язык именуется Visual Basic for Applications, или кратко VBA.

Первая версия языка Visual Basic представляла собой всего лишь интересную игрушку, поскольку разработка серьезных Windows-приложений с его помощью была невозможной. Однако отправная точка для дальнейшей разработки программного продукта была заложена. Каждая последующая версия языка Visual Basic представляла собой очередное усовершенствование предыдущей версии. Именно поэтому в настоящее время Visual Basic является наиболее распространенным языком программирования, применяемым для персональных компьютеров. Согласно оценкам компании Microsoft, свыше трех миллионов человек используют Visual

Basic в той или иной форме, причем половина из них применяет Office VBA, а остальные пользуются VB в чистом виде.

В то время как язык Visual Basic превратился в весьма приличный инструмент, предназначенный для создания самостоятельных Windows-приложений, популярность языка Visual Basic для приложений (VBA) основана на его позиционировании в качестве инструмента программирования для Microsoft Office. В конце концов, основная причина, по которой вы читаете эту книгу, — желание научиться программировать на языке VBA.

Язык программирования высокого уровня VBA лежит в основе ряда важных приложений, в число которых входят Microsoft Word, Excel, Access и PowerPoint, а также Microsoft Visual Basic. Кроме того, другие компании приобретают лицензии на использование языка VBA, а также периодически его применяют для создания собственных приложений.

Любое так называемое *хост-приложение* поддерживает VBA-расширения, что позволяет лучше его приспособить к практическим потребностям. Например, при работе с текстовым процессором Microsoft Word без редактора Word VBA невозможно понять природу заголовков и колонтитулов, процесс проверки грамматики и нумерации страниц и т. п. При пользовании Excel, обрабатывающем электронные таблицы, Excel VBA позволяет понять природу ячеек, формул, диаграмм и т. п. Ну и наконец, в процессе применения редактора Visual Basic, предназначенного для разработки самостоятельных приложений, расширение VBA требуется для управления различными элементами управления (текстовыми полями, командными кнопками, полями со списком).

Языки C и C++

Язык программирования C является потомком ряда более старых языков (включая B). Он был разработан в 1972 году сотрудником лаборатории Bell Laboratories Денисом Ричи (Dennis Ritchie). Этот язык имел достаточно простой синтаксис и включал небольшое количество операторов. Однако он был значительно усовершенствован компанией Microsoft в целях работы в среде PC.

Основным достоинством языка C и его потомка C++ является сочетание преимуществ языка программирования высокого уровня, например приемлемой читаемости кода, с возможностью непосредственного управления операционной системой и аппаратным обеспечением компьютера. К сожалению, эти преимущества могут обернуться недостатками в руках неумелого программиста. Кроме того, синтаксис языка C позволяет всем желающим разрабатывать собственный, как им кажется, краткий и элегантный стиль, который на практике оказывается совершенно нечитаемым.

Для сравнения рассмотрим версию предыдущей программы, вычисляющей среднее арифметическое, но уже на языке C. Я думаю, вы без колебаний согласитесь с тем, что по сравнению с программой на языке BASIC восприятие кода заметно затруднилось. Строки, начинающиеся со знака //, являются комментариями.

```
// Программа на языке C для вычисления среднего арифметического  
// из набора, в который входит максимум 100 чисел
```

```
#include <stdio.h>
```



```
void main(void)
{
    // Описание переменных
    int Num, i;
    float Sum, NextNum, Ave;

    // Запрос на ввод количества чисел
    printf( "Введите количество чисел: " );
    scanf( "%u", &Num );

    // Продолжаем, если параметр Num меньше 100, но больше 1.
    if( (Num > 0) && (Num <= 100) )
    {
        Sum = 0.0;
        // Цикл, накапливающий числа для усреднения
        for( i = 1; i <= Num; i++ )
        {
            // Запрос на ввод следующего числа
            printf( "Введите следующее число: " );
            scanf( "%f", &NextNum );

            // Добавление числа к имеющейся сумме
            Sum += NextNum;
        }

        // Вычисление среднего арифметического
        Ave = Sum / Num;

        // Отображение среднего арифметического
        printf ( "Среднее арифметическое равно: %f\n", Ave );
    }
}
```

Объектно-ориентированное расширение языка C, известное под названием C++, было разработано в начале 80-х годов Бьярни Страуструпом (Bjarne Stroustrup) (также в лаборатории Bell Laboratories).

Visual C++

Вопреки значительному распространению языка Visual Basic он не имеет главенствующего положения в разработке самостоятельных приложений. Эта роль принадлежит языку Visual C++.

Это действительно неплохой язык. Фирма Microsoft выступает против попыток использовать один язык для решения многих разнородных задач. Ведь рост мощности языка Visual Basic (а также VBA), благодаря которому появляется возможность использовать его для написания разнообразных сложных приложений, приводит к неоправданному усложнению. В результате количество пользователей, пишущих программы на этом языке, уменьшается.

Язык Visual C++ является комбинацией языка программирования C++ и графической среды Windows. В отличие от языка Visual Basic Visual C++ не является интуитивно понятным. Частично это обусловлено природой лежащего в основе языка программирования (язык C не так прост для восприятия, как BASIC), частично — тем фактом, что язык C++ является полностью объектно-ориентирован-

ным, а частично потому, что Visual C++ был разработан в целях контроля среды Windows на более фундаментальном уровне, чем Visual Basic. Например, с помощью Visual Basic невозможно создать текстовое поле, текст в котором окрашен различными цветами, или изменить цвет подписи к командной кнопке. Другими словами, при программировании на языке VB (или VBA) мы жертвуем мощностью, получая взамен более простую программную среду.

Pascal

Язык программирования Pascal был разработан Николасом Виртом (Niklaus Wirth) в конце 60-х годов. На тот момент требовался язык, с которым было бы легко работать на различных компьютерах и который мог использоваться студентами в целях моделирования. Другими словами, в язык Pascal включено множество функций, позволяющих создавать хорошо структурированные программы. Во многих университетах Pascal преподается в качестве основного языка программирования. Он также занял свое место на персональных компьютерах, сначала в виде версии Turbo Pascal от фирмы Borland, затем в виде визуальной среды программирования Delphi.

Посмотрим, как программа вычисления среднего арифметического на основе введенных пользователем чисел выглядит на языке Pascal. Текст в фигурных скобках ({,}) представляет собой комментарий.

```
{ Программа на языке Pascal для вычисления среднего на основе набора,  
  в который входит максимум 100 чисел }
```

```
program average (input, output);  
  { Описание переменных }  
  var  
    Num, i : integer;  
    Ave, Sum, NextNum : real;  
begin  
  { Приглашение на ввод количества чисел }  
  writeln('Введите количество чисел');  
  readln(Num);  
  { Если параметр Num больше 1 и меньше 100, продолжаем }  
  if ((Num > 0 ) and (Num <= 100)) then  
    begin  
      Sum := 0;  
      { Цикл, накапливающий числа для усреднения }  
      for i := 1 to Num do  
        begin  
          { Запрос на ввод следующего числа }  
          writeln('Введите следующее число');  
          readln(NextNum);  
          { Добавление числа к имеющейся сумме }  
          Sum := Sum + NextNum;  
        end  
      end  
      { Вычисление среднего арифметического }  
      Ave := Sum / Num;  
      { Отображение среднего арифметического }  
      writeln('Среднее арифметическое: ', Ave);  
    end  
end  
end
```


FORTRAN

Название языка FORTRAN представляет собой сокращение от словосочетания *Formula Translation* (преобразование формул). Этот язык предназначен для решения вычислительных задач, а также является первым широко используемым языком высокого уровня. Его создатель Джон Бэкус (John Backus) утверждал, что его творение не блещет красотой, а просто позволяет избежать ошибок в коде и в последующем процессе отладки!

Вот версия программы для нахождения среднего арифметического на языке FORTRAN. Строки, начинающиеся с буквы C, являются комментариями.

C Программа на языке FORTRAN для вычисления среднего
C арифметического максимум из 100 чисел

```
Real SUM, AVE, NEXTNUM
SUM = 0.0
```

C Приглашение на ввод количества чисел
WRITE(*,*) 'Введите количество чисел: '
READ(*,*) NUM

C Если параметр Num больше 1, но меньше 100, продолжаем

```
IF NUM .GT. 0 .AND. NUM .LE. 100 then
  C Цикл, собирающий числа для усреднения
  DO 10 I = 1, NUM
    C Приглашение на ввод следующего числа
    WRITE(*,*) 'Введите следующее число: '  
    READ(*,*) NEXTNUM
    C Добавление числа к уже имеющейся сумме
    SUM = SUM + NEXTNUM
```

```
10  CONTINUE
  C Вычисление среднего арифметического
  AVE = SUM/NUM
  C Отображение среднего арифметического
  WRITE(*,*) 'Среднее арифметическое: '  
  WRITE(*,*) AVE
ENDIF
```

```
STOP
END
```

COBOL

Название языка COBOL представляет собой аббревиатуру от словосочетания *Common Business Oriented Language* (Общий бизнес-ориентированный язык). Он был разработан в 50-х годах Грейсом Хупером (Grace Hopper) в целях создания бизнес-приложений, для написания которых требовался (по его мнению) английский язык. Создается впечатление, что основной задачей при разработке этого языка было желание избежать любых математических записей. Неизбежной платой за это стало отсутствие лаконичности и низкая степень читаемости кода.

Для иллюстрации рассмотрим код на языке COBOL, который складывает два числа.

* Программа на языке COBOL, складывающая два числа

IDENTIFICATION DIVISION.

PROGRAM-ID. ADD02.

ENVIRONMENT DIVISION.

DATA DIVISION.

WORKING-STORAGE SECTION.

01 FIRST-NUMBER PIC IS 99.

01 SECOND-NUMBER PIC IS 99.

01 SUM PIC IS 999.

PROCEDURE DIVISION.

PROGRAM-BEGIN.

DISPLAY "ВВЕДИТЕ ПЕРВОЕ ЧИСЛО ".

ACCEPT FIRST-NUMBER.

DISPLAY "ВВЕДИТЕ ВТОРОЕ ЧИСЛО ".

ACCEPT SECOND-NUMBER.

COMPUTE SUM = FIRST-NUMBER + SECOND-NUMBER

DISPLAY "СУММА ЧИСЕЛ РАВНА: " SUM.

PROGRAM-DONE.

STOP RUN.

На языке BASIC эта программа выглядела бы так:

INPUT "Введите первое число: ", n1

INPUT "Введите второе число: ", n2

PRINT "Сумма введенных чисел: ", n1 + n2

Этот пример наглядно демонстрирует излишнюю многословность языка COBOL.

LISP

Языки BASIC, C, Pascal и FORTRAN похожи во многих аспектах. Написанные на них программы можно сделать легко читаемыми, особенно если программист ставит перед собой такую задачу. Но существуют и языки, читаемость которых при любых обстоятельствах остается под большим вопросом. К таковым относится разработанный в конце 50-х годов Джоном Мак-Карти (John McCarthy) и Марвином Мински (Marvin Minsky) язык LISP. Он был предназначен для обработки списков в приложениях искусственного интеллекта.

В языке LISP любой элемент представляет собой список. Вот пример кода:

; Пример программы на языке LISP, которая

; берет два списка и возвращает значение

; T (истина), если списки идентичны и значение F в противном случае

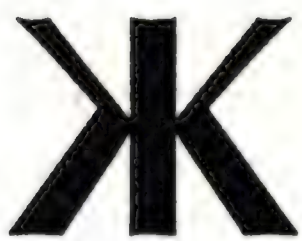
(DEFINE (

'(equal (LAMBDA (list1 list2)

(COND


```
((ATOM list1) (EQ list1 list2))  
((ATOM list1 NIL)  
 (equal (CAR list1) (CAR list2))  
 (equal (CDR list1) (CDR list2)))  
(T NIL)  
)  
))  
))
```

Этот пример доказывает неоспоримый факт: независимо от того, что думают пользователи о компании Microsoft, ее можно по крайней мере поблагодарить за выбор языка BASIC (VBA) в качестве основы для программирования в Microsoft Office!



Новые объекты в Excel XP

Приложение Excel XP содержит 37 новых объектов. Здесь вы найдете их список вместе с маршрутами доступа и кратким описанием каждого из них. Также приводится перечень свойств и методов этих объектов (исключая повсеместно встречающиеся свойства Application, Creator и Parent).

Объект AllowEditRange

Маршрут доступа:

Application -- Range -- Worksheet -- Protection -- AllowEditRanges - AllowEditRange

Объект	Методы	Свойства
Объект AllowEditRange	ChangePassword (пароль как строка)	Range() объект Range
	Delete()	Title() переменная типа String
	Unprotect([Password] переменная типа Variant)	Users() объект UserAccessList
Объект AllowEditRanges	Add(Title переменная типа String, Range переменная типа Range,	Count() переменная типа Long
	[Password] переменная типа Variant) объект AllowEditRange	Item(Index переменная типа Variant) объект AllowEditRange

Объект AllowEditRange защищает от редактирования паролем выделенный диапазон ячеек листа. После защиты диапазона ячеек таким способом *и общей защиты листа* попытки редактирования содержимого ячеек приведут к появлению окна с запросом пароля.

В семействе AllowEditRanges хранятся текущие объекты AllowEditRange.

Объект AutoRecover

Маршрут доступа:
Application - AutoRecover

Объект	Свойства
AutoRecover	Enabled() переменная типа Boolean Path() переменная типа String Time() переменная типа Long

Функция AutoRecover делает возможным восстановление данных при зависании приложения Excel XP. Пользователь может указать папку, в которой будут сохраняться файлы для автовосстановления, а также время между операциями сохранения.

Объект AutoRecover выполняет автоматическое восстановление файлов. Его свойство Path определяет папку, в которой будет сохранен файл автовосстановления, а свойство Time — время (в минутах), прошедшее между созданиями резервных копий файла. Например:

```
Application.AutoRecover.Path = "d:\temp"  
Application.AutoRecover.Time = 10 ' minutes
```

Объект CalculatedMember

Маршрут доступа:
Application -- Range -- PivotTable -- CalculatedMembers - CalculatedMember

Объект	Методы	Свойства
CalculatedMember	Delete()	Formula() переменная типа String IsValid() переменная типа Boolean Name() переменная типа String SolveOrder() переменная типа Long SourceName() переменная типа String Type() перечисление XICalculatedMemberType
CalculatedMembers	Add(Name переменная типа String Formula переменная типа String [SolveOrder] переменная типа Variant [Type] переменная типа Variant) объект CalculatedMember	Count() переменная типа Long Item(Index переменная типа Variant) объект CalculatedMember

Объект CalculatedMember программным способом управляет вычисляемыми полями и вычисляемыми элементами в сводных таблицах, которые связаны с внеш-

ними источниками данных OLAP (Online Analytical Processing — аналитическая обработка в реальном времени). В данной книге эти вопросы рассматриваться не будут.

Объект CellFormat

Маршрут доступа:
Application - CellFormat

Объект	Методы	Свойства
CellFormat	Clear()	AddIndent() переменная типа Variant Borders() переменная типа Borders Font() переменная типа Font FormulaHidden() переменная типа Variant HorizontalAlignment() переменная типа Variant IndentLevel() переменная типа Variant Interior() переменная типа Interior Locked() переменная типа Variant MergeCells() переменная типа Variant NumberFormat() переменная типа Variant NumberFormatLocal() переменная типа Variant Orientation() переменная типа Variant ShrinkToFit() переменная типа Variant VerticalAlignment() переменная типа Variant WrapText() переменная типа Variant

Объект CellFormat используется вместе с объектами FindFormat и ReplaceFormat, что позволяет программным путем находить и изменять форматирование ячеек. Более подробную информацию о нем вы найдете в главе 16.

Объект CustomProperty

Маршрут доступа:
Application -- Range -- Worksheet -- CustomProperties - CustomProperty

Объект	Метод	Свойство
CustomProperty	Delete()	Name() переменная типа String Value() переменная типа Variant
CustomProperties	Add(Name переменная типа String, Value переменная типа Variant) объект CustomProperty	Count() переменная типа Long Item(Index переменная типа Variant) объект CustomProperty

В неразборчивой справочной документации Microsoft об объекте CustomProperty написано буквально следующее: «Представляет идентификатор информации, который может быть использован в качестве метаданных языка XML». Однако в документации ничего не сказано по поводу того, как именно эта информация может быть использована в качестве метаданных. В любом случае описание функций, связанных с языком XML, выходит за рамки данной книги.

Семейство CustomProperties содержит объекты CustomProperty.

Объекты Diagram, DiagramNode и DiagramNodeChildren

Маршрут доступа:
Application -- Chart -- Shapes -- Shape - Diagram -- *DiagramNodes* -- *DiagramNode* - *DiagramNodeChildren*

Объект	Методы	Свойства
Diagram	Convert(Type As MsoDiagramType)	AutoFormat() объект MsoTriState AutoLayout() объект MsoTriState Nodes() объект DiagramNodes Reverse() объект MsoTriState Type() объект MsoDiagramType
DiagramNode	AddNode(pos объект MsoRelativeNodePosition, nodeType объект MsoDiagramNodeType) объект DiagramNode CloneNode(copyChildren как переменную типа Boolean, pTargetNode объект DiagramNode, pos объект MsoRelativeNodePosition) объект DiagramNode Delete() MoveNode(pTargetNode объект DiagramNode, pos объект MsoRelativeNodePosition) NextNode() объект DiagramNode PrevNode() объект DiagramNode ReplaceNode(pTargetNode объект DiagramNode) SwapNode(pTargetNode объект DiagramNode, swapChildren как переменную типа Boolean) TransferChildren(pReceivingNode объект DiagramNode)	Children() объект DiagramNodeChildren Diagram() объект IMsoDiagram Layout() объект MsoOrgChartLayoutType Root() объект DiagramNode Shape() объект Shape TextShape() объект Shape
DiagramNodes	Item(Index переменная типа Variant) объект DiagramNode SelectAll()	Count() как Signed machine int

Объект	Методы	Свойства
DiagramNodeChildren	AddNode(Index переменная типа Variant, nodeType объект MsoDiagramNodeType) объект DiagramNode	Count() как Signed machine int
	Item(Index переменная типа Variant) объект DiagramNode	FirstChild() объект DiagramNode
	SelectAll()	LastChild() объект DiagramNode

Организационные диаграммы являются особым типом форм, который используется в целях большей наглядности представления данных. Всего существует шесть типов организационных диаграмм. Связанные с ними объекты позволяют создавать и редактировать диаграммы программным способом. Более подробную информацию по этой теме можно найти в приложении А.

Объект Error

Маршрут доступа:
Application -- Range -- Errors - Error

Объект	Свойства
Error	Ignore() переменная типа Boolean
	Value() переменная типа Boolean

Объект Error определяет, возникла ли конкретная ошибка в выделенном диапазоне. В частности, свойство Errors объекта Range возвращает семейство Errors. Это семейство содержит точно семь объектов Error, в соответствии с семью типами ошибок, указанных на вкладке Error Checking (Проверка ошибок) диалогового окна Options (Параметры).

Объект Error имеет логическое, предназначенное только для чтения свойство Value, которому присваивается значение True при наличии в рассматриваемом диапазоне ошибок, тип которых определяется объектом Error.

Важно понимать, что свойство Value объекта Error работает корректно только при включенной проверке ошибок определенного типа. Это можно сделать как программным способом, так и установив на вкладке Error Checking (Проверка ошибок) соответствующий флажок.

Для иллюстрации рассмотрим код, включающий изначально глобальную проверку наличия пустых ячеек. Затем он помещает в ячейку A1 формулу, которая ссылается на одну или несколько пустых ячеек. Напоследок код проверяет, существует ли ошибка, связанная с пустыми ячейками в диапазоне, определенном переменной rng.

```
Sub Example_ErrorObject()  
  
Dim rng As Range  
Set rng = Application.Range("A1")  
  
' Убедимся, что проверка ошибки, связанной с пустыми ячейками, включена
```



```
Application.ErrorCheckingOptions.EmptyCellReferences = True

' Включим формулу со ссылкой на пустые ячейки
' в ячейку диапазона rng
Range("A1").Formula = "=A12+A13"

' Проверим, есть ли в указанном диапазоне ошибки, связанные с пустыми
' ячейками
If rng.Errors.Item(xlEmptyCellReferences).Value = True Then
    MsgBox "Формула ссылается на пустые ячейки " & rng.Address
Else
    MsgBox "Формула не ссылается на пустые ячейки " & rng.Address
End If

End Sub
```

Обратите внимание на то, что если поменять указанный диапазон на диапазон, содержащий более одной ячейки:

```
Set rng = Application.Range("A1:B2")
```

строка:

```
If rng.Errors.Item(xlEmptyCellReferences).Value = True Then
```

будет приводить к появлению общего (и, следовательно, совершенно бесполезного) сообщения об ошибке! При этом непонятно, почему свойство `Errors` объекта `Range` не может работать с диапазонами, содержащими более одной ячейки. Для этого нет совершенно никаких причин, а в справочной документации нет упоминания о подобных ограничениях. К сожалению, невозможно определить, является ли это пропуском в документации или же ошибкой в коде Excel. Единственной приходящей на ум причиной может являться тот факт, что объект `Error` не включает свойства, возвращающие список ячеек с ошибками, расположенных в определенном диапазоне. Это заставляет поверить, что свойство `Errors` изначально предназначалось для работы с диапазоном, состоящим всего из одной ячейки.

Объект ErrorCheckingOptions

Маршрут доступа:

```
Application - ErrorCheckingOptions
```

Объект	Свойства
ErrorCheckingOptions	BackgroundChecking() переменная типа Boolean EmptyCellReferences() переменная типа Boolean EvaluateToError() переменная типа Boolean InconsistentFormula() переменная типа Boolean IndicatorColorIndex() перечисление <code>xlColorIndex</code> NumberAsText() переменная типа Boolean OmittedCells() переменная типа Boolean TextDate() переменная типа Boolean UnlockedFormulaCells() переменная типа Boolean

Объект `ErrorCheckingOptions` соответствует вкладке `Error Checking` (Проверка ошибок) диалогового окна `Options` (Параметры), а его свойства соответствуют флажкам из раздела `Rules` (Проверять наличие следующих ошибок), как показано на рис. Ж.1.

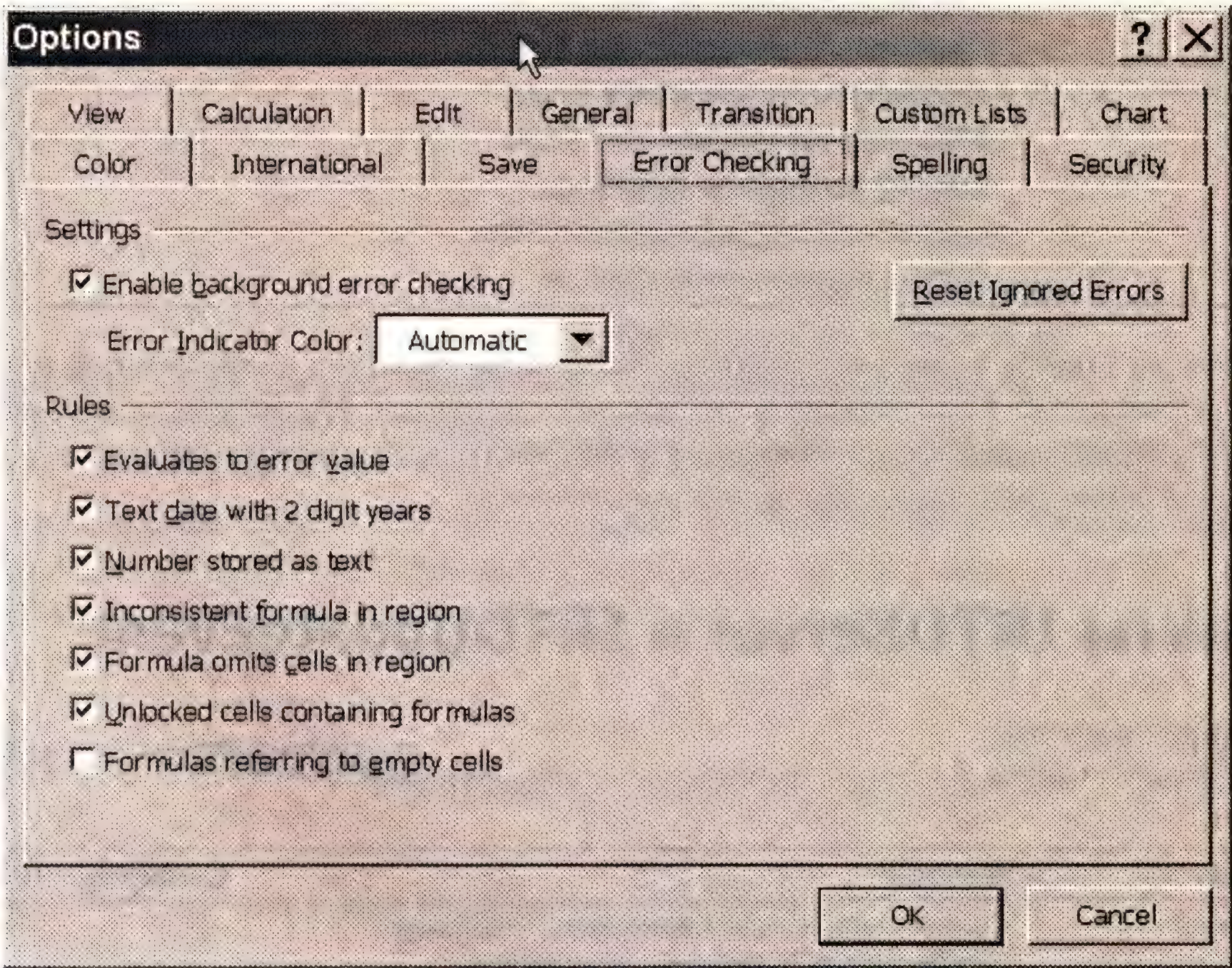


Рис. Ж.1. Вкладка `Error Checking`

Например, если логическое свойство `TextDate` объекта `ErrorCheckingOptions` имеет значение `True` (это соответствует установке флажка `Text date with 2 digit years` (Текстовая дата с 2-значным годом)), рядом с ячейкой, содержащей год в виде двухзначного вместо четырехзначного числа, отобразится кнопка `AutoCorrect`.

Семейство `Errors` содержит объекты `Error`. При этом оно имеет свойство `Item(Index переменная типа Variant)` объекта `Error`.

Объект `Graphic`

Маршрут доступа:
`Application -- Chart -- PageSetup - Graphic`

Объект	Свойство
Graphic	<code>Brightness()</code> переменная типа <code>Single</code>
	<code>ColorType()</code> объект <code>MsoPictureColorType</code>
	<code>Contrast()</code> переменная типа <code>Single</code>

продолжение ➤

Продолжение таблицы

Объект	Свойство
	CropBottom() переменная типа Single
	CropLeft() переменная типа Single
	CropRight() переменная типа Single
	CropTop() переменная типа Single
	Filename() переменная типа Single
	Height() переменная типа Single
	LockAspectRatio() объект MsoTriState
	Width() переменная типа Single

Объект Graphic позволяет задавать различные свойства изображений, которые могут появляться в заголовках и нижних колонтитулах. Например, можно определить имя файла, размер изображения и его различные цветовые аспекты, а также его расположение в заголовке или в колонтитуле.

Объекты IRTDServer и IRTDUpdateEvent

Маршрут доступа:
Application -- IRtdServer; Path: Application - IRTDUpdateEvent

Объект	Методы	Свойства
IrtdServer	ConnectData(TopicID переменная типа Long, Strings массив пременных типа Variant, GetNewValues переменная типа Boolean) переменная типа Variant DisconnectData(TopicID как параметр типа Long) Heartbeat() переменная типа Long RefreshData(TopicCount переменная типа Long) массив переменных типа Variant ServerStart(CallbackObject объект IRTDUpdateEvent) переменная типа Long ServerTerminate()	
IRTDUpdateEvent	Disconnect() UpdateNotify()	HeartbeatInterval() переменная типа Long

Объекты IRTDServer и IRTDUpdateEvent относятся к серверам данных, работающим в реальном времени, и рассказ о них выходит за рамки данной книги.

Объекты PivotCell и PivotItemList

Маршрут доступа:
Application -- Range - PivotCell - PivotItemList

Объект	Методы	Свойства
PivotCell		ColumnItems() объект PivotItemList CustomSubtotalFunction() перечисление XlConsolidationFunction DataField() объект PivotField PivotCellType() перечисление XlPivotCellType PivotField() объект PivotField PivotItem() объект PivotItem PivotTable() объект PivotTable Range() переменная типа Range RowItems() объект PivotItemList
PivotItemList	Item(Index переменная типа Variant) объект PivotItem	Count() переменная типа Long

В приложении Excel XP впервые появился объект PivotCell, представляющий собой ячейку сводной таблицы. Дополнительную информацию по этому поводу можно найти в главе 20.

Объект Protection

Маршрут доступа:
Application -- Range -- Worksheet - Protection

Объект	Свойства
Protection	AllowDeletingColumns() переменная типа Boolean AllowDeletingRows() переменная типа Boolean AllowEditRanges() объект AllowEditRanges AllowFiltering() переменная типа Boolean AllowFormattingCells() переменная типа Boolean AllowFormattingColumns() переменная типа Boolean AllowFormattingRows() переменная типа Boolean AllowInsertingColumns() переменная типа Boolean AllowInsertingHyperlinks() переменная типа Boolean AllowInsertingRows() переменная типа Boolean AllowSorting() переменная типа Boolean AllowUsingPivotTables() переменная типа Boolean

При защите листа приложение позволяет указывать список операций, которые разрешены для выполнения в незаблокированных ячейках. На пользовательском уровне это делается в диалоговом окне Protection (Защита листа). На уровне программного кода для этого требуется воспользоваться свойствами объекта Protection.

Объект RTD

Маршрут доступа:
Application - RTD

Объект	Методы	Свойства
RTD	RefreshData() RestartServers()	ThrottleInterval() переменная типа Long

Объект RTD представляет собой объект данных режима реального времени, рассказ о котором выходит за рамки данной книги.

Объекты, связанные со смарт-тегами

Маршрут доступа:
Application -- Range -- SmartTags - SmartTag --SmartTagActions -- SmartTagAction;

Маршрут доступа:
Application -- Workbook -- SmartTagOptions;

Маршрут доступа:
Application -- SmartTagRecognizers - SmartTagRecognizer

Объект	Методы	Свойства
SmartTag	Delete()	DownloadURL() переменная типа String Name() переменная типа String Properties() объект CustomProperties Range() переменная типа Range SmartTagActions() объект SmartTagActions XML() переменная типа String
SmartTags	Add(SmartTagType переменная типа String) объект SmartTag	Count() переменная типа Long
SmartTagAction	Execute()	Name() переменная типа String
SmartTagActions		Count() переменная типа Long Item(Index переменная типа Variant) объект SmartTagAction
SmartTagOptions		DisplaySmartTags() перечисление XISmartTagDisplayMode EmbedSmartTags() переменная типа Boolean
SmartTagRecognizer		Enabled() переменная типа Boolean FullName() переменная типа String progID() переменная типа String
SmartTagRecognizers		Count() переменная типа Long Item(Index переменная типа Variant) объект SmartTagRecognizer Recognize() переменная типа Boolean

Этот набор объектов обеспечивает программный контроль над существующими смарт-тегами. Для получения пользовательских смарт-тегов требуется приложение, которое может создавать серверы автоматизации (COM DLL), то есть редактор Visual Basic или Visual C++. В Excel XP это сделать невозможно. Более подробную информацию по данной теме можно найти в главе 22.

Объект Speech

Маршрут доступа:
Application - Speech

Объект	Методы	Свойства
Speech	Speak(Text переменная типа String, [SpeakAsync] переменная типа Variant, [SpeakXML] переменная типа Variant, [Purge] переменная типа Variant)	Direction() объект XISpeakDirection SpeakCellOnEnter() переменная типа Boolean

Объект Speech используется для внедрения в Excel XP речевых сигналов. Для его получения применяется свойство Speech объекта Application. Этот объект обладает только двумя свойствами — Direction и SpeakCellOnEnter и единственным методом — Speak.

Свойство Direction определяет направление передачи речевых сигналов и может принимать значения констант из следующего перечисления:

```
Enum XISpeakDirection
    xISpeakByRows = 0
    xISpeakByColumns = 1
End Enum
```

В данном случае направление передачи речевых сигналов задается с помощью свойства Speak объекта Range. Например, если в активных ячейках содержится следующая информация:

```
A1 = 1, B1 = 2
A2 = 3, B2 = 4
```

то код:

```
Range("A1:B2").Speak xISpeakByRows
```

приведет к следующему порядку воспроизведения данных: 1, 2, 3, 4, в то время как код:

```
Range("A1:B2").Speak xISpeakByColumns
```

воспроизведет данные так: 1, 3, 2, 4.

Логическое свойство SpeakCellOnEnter указывает на то, будет ли Excel воспроизводить данные, содержащиеся в активных ячейках при нажатии клавиши Enter или при изменении активной ячейки. При переходе к другой ячейке воспроизведение данных, содержащихся в ранее активной ячейке, происходит только при условии редактирования этих данных.

Метод `Speak` приводит к воспроизведению текста:

```
Application.Speech.Speak "Быть или не быть?"
```

или:

```
Application.Speech.Speak ActiveSheet.Range("A1").Value
```

Объект SpellingOptions

Маршрут доступа:
`Application - SpellingOptions`

Объект	Свойство
Spelling Options	<code>ArabicModes()</code> перечисление <code>XlArabicModes</code>
	<code>DictLang()</code> переменная типа <code>Long</code>
	<code>GermanPostReform()</code> переменная типа <code>Boolean</code>
	<code>HebrewModes()</code> перечисление <code>XlHebrewModes</code>
	<code>IgnoreCaps()</code> переменная типа <code>Boolean</code>
	<code>IgnoreFileNames()</code> переменная типа <code>Boolean</code>
	<code>IgnoreMixedDigits()</code> переменная типа <code>Boolean</code>
	<code>KoreanCombineAux()</code> переменная типа <code>Boolean</code>
	<code>KoreanProcessCompound()</code> переменная типа <code>Boolean</code>
	<code>KoreanUseAutoChangeList()</code> переменная типа <code>Boolean</code>
	<code>SuggestMainOnly()</code> переменная типа <code>Boolean</code>
	<code>UserDict()</code> переменная типа <code>String</code>

Объект `SpellingOptions` возвращается свойством `SpellingOptions` объекта `Application` и задает параметры блока орфографического контроля. Этот объект не имеет методов, а его свойства по большей части соответствуют флажкам на вкладке `Spelling` (Орфография) диалогового окна `Options` (Параметры).

Объект Tab

Маршрут доступа:
`Application -- Chart - Tab`

Объект	Свойства
Tab	<code>Color()</code> переменная типа <code>Variant</code>
	<code>ColorIndex()</code> перечисление <code>XlColorIndex</code>

Возможность изменять цвет листа и ярлыков диаграммы появилась только в Excel XP. Для изменения цвета следует написать:

```
Worksheets("Sheet2").Tab.Color = vbGreen
```

Объект `Tab` возвращается свойством `Tab` объектов `Worksheet` и `Chart` и имеет свойства `Color` и `ColorIndex`, которые определяют цвет.

Объект UsedObjects

Маршрут доступа:
Application - UsedObjects

Объект	Свойства
UsedObjects	Count() переменная типа Long Item(Index As Variant) переменная типа Object

В справочной документации информация о семействе UsedObjects отражена крайне скудно, даже по стандартам компании Microsoft. Совершенно непонятно, что такое использованные объекты, зато рассказывается, как сосчитать их число!

Если открыть новую книгу (с тремя появляющимися по умолчанию пустыми листами), тогда код:

Application.UsedObjects.Count

вернет число 7. Соответственно, чем бы ни являлись использованные объекты, новая книга содержит их в количестве семи штук.

После ряда экспериментов я заключил, что если компания Microsoft не посчитала необходимым потратить время на написание хотя бы одного предложения по поводу данного объекта, вряд ли стоит тратить мое и ваше время на разгадку этого ребуса.

Объекты UserAccessList и UserAccess

Маршрут доступа:
Application -- Range -- Worksheet -- Protection -- AllowEditRanges -- AllowEditRange -- UserAccessList - UserAccess

Объект	Методы	Свойства
UserAccess	Delete()	AllowEdit() переменная типа Boolean Name() переменная типа String
UserAccessList	Add(Name переменная типа String, AllowEdit переменная типа Boolean) объект UserAccess DeleteAll()	Count() переменная типа Long Item(Index переменная типа Variant) объект UserAccess

Объект UserAccess определяет список пользователей, у которых имеется доступ к защищенному диапазону без пароля.

Объект Watch

Маршрут доступа:
Application -- Watches - Watch

Объект	Методы	Свойства
Watch	Delete()	Source() переменная типа Variant
Watches	Add(Source переменная типа Variant) объект Watch Delete()	Count() переменная типа Long Item(Index переменная типа Variant) объект Watch

В Excel XP панель инструментов окна Watch является очень простым, но крайне полезным устройством. Она позволяет пользователю видеть содержимое ячеек, даже если часть листа, на котором они находятся, в данный момент невидима. Инструмент Watch Window показан на рис. Ж.2.

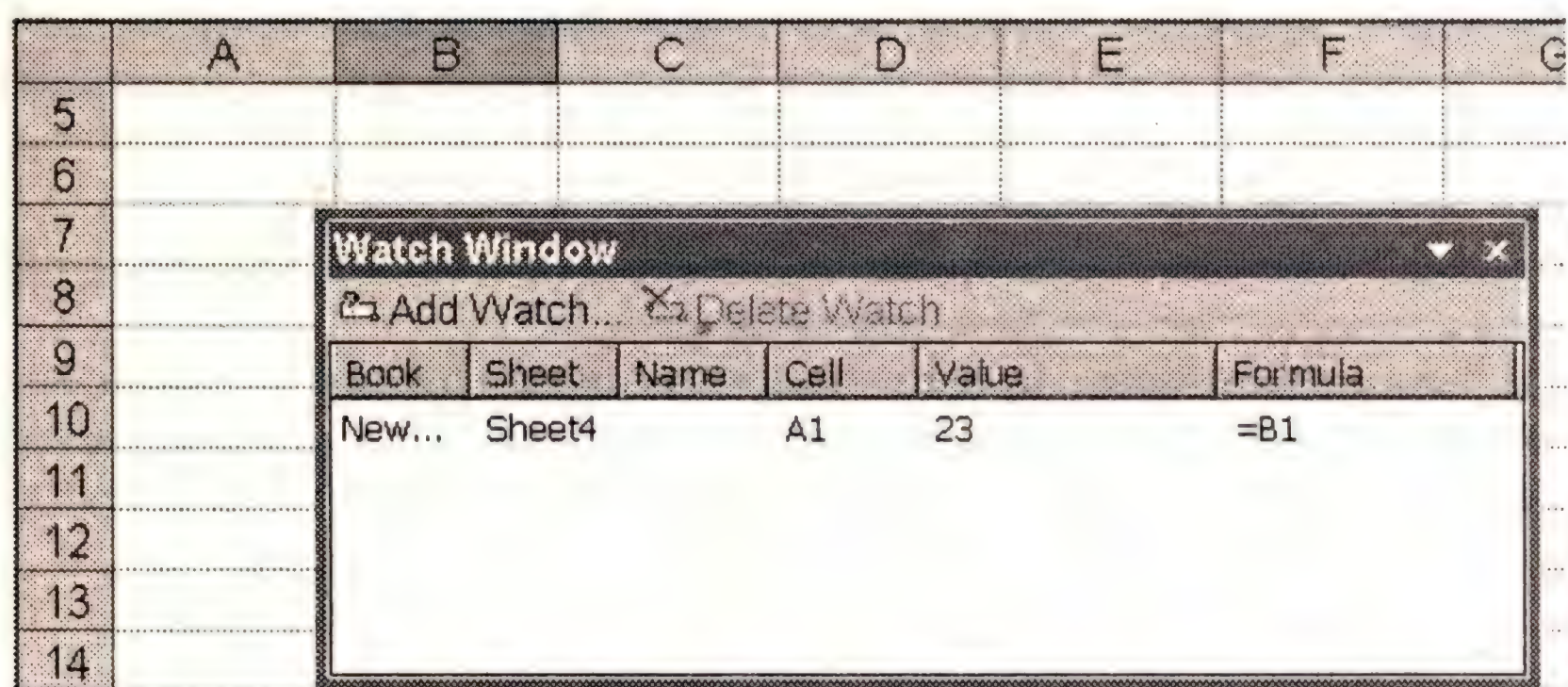


Рис. Ж.2. Инструмент Watch Window

Объект Watch программным образом добавляет инструмент Watch Window.

Свойство Watches объекта Application возвращает семейство Watches, содержащее все открытые объекты Watch. Сам по себе объект Watch имеет только одно реально полезное свойство — Source, которое указывает просматриваемые ячейки. Он имеет также только один метод — Delete, который удаляет окно просмотра.

Например, окно, показанное на рис. Ж.2, может быть создано любым из следующих способов:

```
Dim wa As Watch
Set wa = Application.Watches.Add("A1")
Set wa = Application.Watches.Add(Range("A1"))
```


Алфавитный указатель

A

ActiveWorkbook, 214
Add
 Watch, 57
Add-Ins, 128
Additional Controls, 169
AddToMru, 218
AddToTable, 324
After, 236, 374
Application, 98
Application modal dialog box, 90
Area, 380
Argument, 83
ArrangeStyle, 214
«Array», 71
Assign Macro, 172
Auto Data Tip, 57
AutoFilter, 235
Automatic, 355
AutomaticStyles, 241
Autotemplate, 128
AxisGroup, 385

B

Bar, 380
BasedOn, 230
Before, 236, 374
Bookmark, 47
«Boolean», 66, 68
Break
 When Value Changes, 58
 When Value is True, 57

Breakpoint, 49
Bubble, 380
By reference, 85
By value, 85
ByRef, 85
«Byte», 66
ByVal, 85

C

Call, 83
CategoryLabels, 407, 415
CategoryTitle, 415
ChartType, 383
Child object, 111
Code, 40, 46
Collate, 226, 237, 280, 417
Collection object, 111
Column, 380, 381
ColumnGrand, 318
CommandButton, 174
Compile, 129
 error, 51
 Project, 51
Cone, 381
ConsecutiveDelimiter, 219, 283
Content, 237
Control Toolbox, 171
Copies, 226, 237
Count, 374
Create, 59
CreateBackup, 227
«Currency», 66, 68

Customize, 123
Cylinder, 381

D

«Data», 69
DataType, 218
«Date», 66
Debug, 49
Debugging, 53
Declarations, 42
Default, 209
Design Mode, 172
«Dim», 66
Do, 105
Docking, 45
«Double», 66, 68
Doughnut, 381
DrawingObject, 237

E

EarleistTime, 197
Edit, 59
 mode, 125
Editor, 41
«Empty», 79
«Enum», 63
Error
 handling code, 99
Event code, 136
Expression, 57

F

Field, 341
FieldInfo, 219, 284
FileFilter, 200
FileName, 218, 416
FilterIndex, 201
FilterName, 416
For Each, 104
Format, 236, 415
Forms, 172
From, 226, 237
FromReferenceStyle, 205
Full-module view, 40
Function, 48, 82

G

Gallery, 415
General, 41

H

HasAutoFormat, 318
HasLegend, 415
Height, 375
HelpContext, 210
HelpFile, 210
Hide, 126

I

Immediate, 43
In-line error checking, 100
InitialFilename, 201
«Integer», 66, 68
Interactive, 416

L

Label, 98
LatestTime, 197
Left, 209, 375
LegendKey, 409
length, 388
Line, 381
LISP, 479
ListBox, 173
Locals, 57
Lock Project for Viewing, 129
«Long», 66, 68

M

Members, 111
Method, 111
Mode, 332

N

Name, 48
Named arguments, 84
New
 Menu, 125
 Toolbar, 124
Nothing, 118

«Nothing», 71, 79

Null, 96

O

«Object», 66

Object Browser, 186

Optional, 83

Options, 41

Order, 341

Origin, 218

P

PageFieldOrder, 318

PageFieldWrapCount, 318

Parameter, 83

Parent object, 111

Password, 218, 237

PasteSpecial, 236

Pie, 381

Pivot tables, 311

PlotBy, 415

Positional arguments, 84

Preview, 226

PreviousSelections, 207

PrintToFile, 226, 237, 280, 417

Private, 87

«Private», 76

Procedure, 41, 48

Procedure view, 40

Project/Library, 187

Prompt, 209

Properties, 39

Property, 48, 110

Public, 87

«Public», 76

Pyramid, 381

Q

Qualified procedure name, 88

Quick Watch, 56

R

Radar, 381

«Range», 71

ReadOnly, 218

ReadOnlyRecommended, 227

Record New Macro, 59

RefEdit, 425, 427

Reference, 207

References, 88

RelativeTo, 205

Replace, 238, 407

Reset, 49

Rowcol, 407

RowGrand, 318

Run, 48

mode, 125

to Cursor, 56

S

SaveChanges, 224

Scenarios, 237

Schedule, 198

Scroll, 207

Search Text, 187

Select, 207

Semiautomatic, 198

SeriesLabels, 407, 415

Set

Next Statement, 56

SetPreview, 237, 417

«Single», 66, 68

SourceData, 318

SourceType, 317

start, 388

StartRow, 218

Step, 104

Into, 55

Out, 55

Over, 55

Stock Quotes, 382

«String», 66

Structure, 227

StyleName, 229

Sub, 48, 82

Summary, 129

Surface, 382

SyncHorizontal, 214

System modal dialog box, 90

T

TableDestination, 318
 TableName, 318
 Template, 216
 TextQualifier, 219, 283
 Timer, 169
 Title, 201, 209, 415
 To, 226, 237
 ToAbsolute, 205
 Toggle Breakpoint, 49
 Toolbars, 123
 Toolbox, 38, 168
 Top, 209, 375
 ToReferenceStyle, 205
 Type, 48, 210

U

Union, 211
 Until, 105
 «User-Defined», 66
 UserInterfaceOnly, 237

V

ValueTitle, 415
 «Variant», 66
 VbMsgBoxResult, 91
 Visual Basic Editor, 35

W

Watch
 Expression, 57
 Type, 57
 Watch Window, 494
 Watches, 57
 While, 105
 Width, 375
 Window, 227
 WriteResPassword, 227

X

xlDisabled, 195
 xlDisplayShapes, 224
 xlErrorHandler, 195
 xlHide, 224
 xlInterrupt, 195

xlOff, 203
 xlOn, 203
 xlPlaceholders, 224
 xlStrict, 203
 xlWait, 202
 XY Scatter, 382

Z

z-порядок, 436

A

Автоматизация, 467
 Автошаблон, 128
 Аргумент, 83
 именованный, 84
 позиционный, 84

Б

Биржевая диаграмма, 382
 Браузер объектов, 186

В

Вторичные оси, 385
 Вычисляемое поле, 364

Г

Гистограмма, 380, 381
 График, 381
 Группировка, 344
 Групповой символ, 95

Д

Диаграмма
 с областями, 380
 Дочерний объект, 111

З

Загрузочная папка, 126
 Закладка, 47

И

Именнованные
 поля данных, 323
 Имя проекта, 37

К

Клавиатурная комбинация, 46
Кнопка, 144
Код
 события, 136
Код обработки ошибок, 99
Кольцевая диаграмма, 381
«Комментарии», 30, 62
Коническая диаграмма, 381
«Константа», 63
Контекстная проверка ошибок, 100
Контроллер автоматизации, 467
Контрольное значение, 56
Круговая диаграмма, 381

Л

Легенда, 403
Лепестковая диаграмма, 381
Линейчатая диаграмма, 380
Лист диаграммы, 372

М

Макросы, 58
«Массив», 71
Мастер сводных таблиц, 314
Метод, 111
 Activate, 136, 224, 235
 Add, 112, 216, 229
 AddDiagram, 447
 AddFields, 324
 AddItem, 177
 ApplyCustomType, 382
 ApplyDataLabels, 409
 Arrange, 214
 AutoShow, 340
 AutoSort, 341
 BringToFront, 379, 438
 Calculate, 198, 235
 CalculatedFields, 335
 CalculateFullRebuild, 198
 ChartWizard, 414
 CheckAbort, 199
 Clear, 99, 209
 Close, 112, 217, 224
 ConvertFormula, 205

Метод (*продолжение*)

Copy, 235
CreateNames, 213
Delete, 112, 213, 236
Evaluate, 206, 236
Execute, 434
Export, 416
FindControl, 149
FindFile, 200
GetOpenFilename, 200
GetSaveAsFilename, 201
Goto, 207
Group, 241, 344
InchesToPoints, 375
InputBox, 209
Intersect, 211
Item, 112
ListFormulas, 335, 367
Move, 230, 236
NewWindow, 214
OnKey, 196
OnTime, 197
Open, 218
OpenText, 218
PasteSpecial, 236
PivotItems, 357
PivotSelect, 331
PivotTableWizard, 316
PrintOut, 112, 226, 237, 416
PrintPreview, 226, 237
Protect, 111, 226, 237
Quit, 208
Refresh, 355
RefreshAll, 227
RefreshTable, 337
Remove, 112
Save, 227
SaveAs, 227, 238
SaveCopyAs, 228
SaveWorkspace, 202
Select, 238
SendToBack, 379, 438
SetBackgroundPicture, 239
Show, 165
ShowDataForm, 239
ShowLevels, 242

Метод (*продолжение*)

ShowPages, 338
 Subtotals, 353
 Ungroup, 344
 Unprotect, 239
 ZOrder, 436

Модальное диалоговое окно, 90

Модуль, 38, 41

класса, 38

«Модульность», 32

Н

Надстройка, 128

Необязательный аргумент, 83

О

Область, 286

Объект

AllowEditRange, 244, 481
 AutoRecover, 482
 AutoShape, 438
 Axis, 386
 AxisTitle, 387
 Border, 388
 CalculatedMember, 482
 CellFormat, 208, 483
 Characters, 388
 Chart, 372
 ChartArea, 397
 ChartFillFormat, 388, 397
 ChartGroup, 397
 ChartObject, 372
 ChartTitle, 401
 Collection, 111
 CommandBar, 142
 CommandBarControl, 143
 CustomProperty, 484
 DataLabel, 408
 DataTable, 402
 Diagram, 446, 484
 DiagramNode, 446, 484
 DiagramNodeChildren, 447, 484
 Dialog, 162
 DisplayUnitLabel, 391
 DownBars, 399

Объект (*продолжение*)

DropLines, 400
 Error, 99, 485
 ErrorCheckingOptions, 487
 FillFormat, 441
 Floor, 403
 Font, 388
 Global, 119
 Graphic, 488
 HiLoLines, 400
 Interior, 388, 397
 IRTDServer, 488
 IRTDUpdateEvent, 488
 Legend, 403
 LegendEntry, 404
 LegendKey, 404
 Name, 212
 OLEObject, 177
 Outline, 241
 PageSetup, 242, 405
 PivotCache, 355
 PivotCell, 361, 488
 PivotField, 338
 PivotItem, 356
 PivotTable, 319
 PlotArea, 405
 Point, 410
 Protection, 243, 489
 Range, 112
 RTD, 490
 Series, 406
 SeriesLines, 400
 Shape, 436
 ShapeRange, 436
 SmartTag, 433, 490
 SmartTagAction, 433, 490
 SmartTagOptions, 434, 490
 SmartTagRecognizer, 432, 490
 Speech, 491
 SpellingOptions, 492
 Style, 229
 Tab, 492
 TextFrame, 441
 ThisWorkbook, 37
 TickLabels, 393

Объект (*продолжение*)

- UpBars, 399
- UserAccess, 245, 493
- UserForm, 38, 167
- Watch, 494
- Workbook, 221
- Worksheet, 233
- WorksheetFunction, 215

Объектная

- переменная, 115

Объектная модель

- иерархия, 113

Окно

- проекта, 35

Оператор, 79

- Beep, 98
- Const, 63
- GOTO, 30
- End Function, 48
- End Sub, 48
- Exit
 - Do, 105
 - For, 103
 - Function, 87
 - Sub, 87
- For...Next, 103
- If, 65
- If...Then, 102
- Is, 118
- Like, 94
- Mod, 79
- Next, 103
- On Error, 98
- Open, 107
- Option
 - Compare, 94
 - Explicit, 68
- Preserve, 72
- ReDim, 72
- Resume, 100
- Select Case, 106
- Set, 71
- Stop, 49
- Watch, 49
- With, 116

- Описание функции, 81

- Организационные диаграммы, 445

- Отладка, 53

Ошибка

- логическая, 52
- компиляции, 51

П

- Параметр, 83

- глобальный, 119

- Первичные оси, 385

- Передача аргумента

- по значению, 85
- по ссылке, 85

- «Переменная

- статическая», 77

- «Переменная», 65

- Переход к курсору, 56

- «Перечисление», 63

- Перечисление

- VbMsgBoxStyle, 90

- Перечисления, 111

- Пирамидальная диаграмма, 381

- Поверхность, 382

- «Подпрограмма», 33

- Представление

- полного модуля, 40
- процедуры, 40

- «Процедура», 34

- Процедура

- создание, 48

- Пузырьковая диаграмма, 380

Р

- Разделитель текста, 219, 283

- Раскрывающиеся меню, 144

- редактор см. AppBrowser

- Редактор Visual Basic, 35

Режим

- выполнения, 48, 125
- конструирования, 172
- прерывания, 49
- редактирования, 125

- Режимное диалоговое окно, 90

- Родительский объект, 111

С

Сводные таблицы, 311

Свойство, 110

ActiveChart, 111

ActiveSheet, 222

AllowSorting, 244

Application, 113

AutoFilterMode, 235

AutoFormatAsYouType

ReplaceHyperlinks, 195

AutoScaleFont, 387

AutoShowCount, 340

AutoShowField, 341

AutoShowRange, 341

AutoShowType, 341

AxisGroup, 387

AxisTitle, 387

BaseField, 346

BaseItem, 346

BaseUnit, 390

Border, 388

BottomRightCell, 179

CalculateBeforeSave, 199

Calculation, 111, 198, 345

Cancel, 247, 369, 421

Caption, 151, 387

CategoryNames, 389

CategoryType, 390

Cells, 99, 194

Characters, 387

ChartObjects, 375

Charts, 223, 374

ChartType, 379

CodeName, 235

Color, 389

ColorIndex, 389

ColumnFields, 325

ColumnGrand, 327

ColumnItems, 361

ColumnRange, 327

Columns, 194

Command, 143

CommandBars, 145

Controls, 149

CopyObjectsWithCells, 202

Свойство (*продолжение*)

Count, 99, 112, 217

Crosses, 390

CrossesAt, 391

CurrentPage, 351

Cursor, 202

CustomSubtotalFunction, 363

CutCopyMode, 202

DataBodyRange, 329

DataEntryMode, 203

DataField, 362

DataFields, 326

DataLabelRange, 330

DataRange, 343, 357

DataType, 351

DefaultFilePath, 199

DefaultSaveFormat, 199

Description, 99

Direction, 499

DisplayAlert, 194

DisplayAlerts, 208

DisplayCommentIndicator, 194

DisplayDrawingObjects, 224

DisplayErrorString, 335

DisplayFormulaBar, 194

DisplayFullScreen, 194

DisplayNoteIndicator, 194

DisplayNullString, 336

DisplaySmartTags, 434

DisplayUnit, 391

DisplayUnitCustom, 391

DisplayUnitLabel, 391

DownloadURL, 433

DragToColumn, 343

DragToHide, 344

DragToPage, 344

DragToRow, 344

EditDirectlyInCell, 203

EmbedSmartTags, 434

EnableAnimations, 195

EnableAutoComplete, 195

EnableCalculation, 236

EnableCancelKey, 195

Enabled, 174, 432

EnableDrillDown, 336

EnableEvents, 136, 195

Свойство (продолжение)

EnableRefresh, 355
EnableSound, 195
ErrorString, 336
Explosion, 410
FileDialog, 200
FileFormat, 224
Fill, 388
FilterMode, 236
FindFormat, 208
FixedDecimal, 203
FixedDecimalPlaces, 203
Font, 388
FullName, 225, 432
Function, 342
HasAutoFormat, 337
HasAxis, 385
HasDataTable, 402
HasDisplayUnitLabel, 391
HasDropLines, 400
HasHiLoLines, 400
HasLegend, 404
HasMajorGridlines, 392
HasMinorGridlines, 392
HasPassword, 226
HasSeriesLines, 400
HasTitle, 387
Height, 392
HiddenFields, 326
HiddenItems, 351
HorizontalAlignment, 388
Interactive, 203
Interior, 388
IsAddIn, 131
IsCalculated, 358, 367
Item, 217
LabelRange, 343, 358
Left, 393
LineStyle, 389
LinkedCell, 179
ListFillRange, 179
ListIndex, 175
MajorGridlines, 392
MajorTickMark, 393
MajorUnit, 394
MajorUnitIsAuto, 394

Свойство (продолжение)

MajorUnitScale, 394
ManualUpdate, 337
MarkerSize, 411
MarkerStyle, 411
Maximum, 202
MaximumScale, 394
MaximumScaleIsAuto, 394
MemoryUsed, 352, 356
MergeLabels, 337
MinimumScale, 394
MinimumScaleIsAuto, 394
MinorGridlines, 392
MinorScaleUnit, 394
MinorTickMark, 393
MinorUnit, 394
MinorUnitIsAuto, 394
MoveAfterReturn, 203
MoveAfterReturnDirection, 203
MultiSelect, 201, 246, 368, 421
Name, 225, 338
NullString, 336
Number, 99
NumberFormat, 342, 388
OLEObject, 177
OnAction, 144
OptimizeCache, 356
Orientation, 320, 343, 388
Outline, 241
PageFieldOrder, 337
PageFields, 326
PageFieldStyle, 338
PageFieldWrapCount, 338
PageRange, 330
PageRangeCells, 330
PageSetup, 242
Path, 225
PivotCell, 361
PivotCellType, 363
PivotField, 363
PivotFields, 326
PivotItem, 363
PivotSelection, 335
PivotTable, 363
Placement, 179
Position, 343, 358

Свойство (*продолжение*)

PrecisionAsDisplayed, 226
 PreserveFormatting, 337
 PrintObject, 179
 PrintTitleRows, 243
 progID, 432
 ProtectionMode, 238
 ReadOnly, 227
 RecentFiles, 201
 RecordCount, 356, 359
 ReferenceStyle, 204
 RefersTo, 213
 RefersToR1C1, 213
 RefersToRange, 213
 RefreshDate, 337, 355
 RefreshName, 337, 355
 RefreshOnFileOpen, 355
 ReplaceFormat, 208
 ReversePlotOrder, 395
 RowFields, 326
 RowGrand, 327
 RowItems, 361
 RowRange, 330
 Rows, 194
 Saved, 208, 224, 228
 SaveData, 338
 ScaleType, 395
 ScreenUpdating, 204
 ScrollArea, 238
 Selection, 119, 211
 ServerBased, 352
 Sheets, 223
 SheetsInNewWorkbook, 204, 217
 ShowAllItems, 352
 ShowChartTipNames, 204
 ShowChartTipValues, 204
 ShowDetail, 359
 ShowLegendKey, 403
 ShowToolTips, 204
 SmartTagOptions, 434
 SmartTags, 433
 Source, 99
 SourceData, 338, 356
 SourceName, 344
 SpeakCellOnEnter, 499
 Sql, 356

Свойство (*продолжение*)

StandardFont, 204
 StandardFontSize, 204
 StartupPath, 204
 StatusBar, 98, 211
 Subtotals, 327
 SummaryColumn, 241
 SummaryRow, 241
 TabIndex, 173, 246, 368, 421
 TableRange1, 330
 TableRange2, 330
 TableStyle, 337
 TemplatesPath, 204
 TextDate, 495
 ThisWorkbook, 193
 TickLabelPosition, 393
 TickLabels, 393
 TickLabelSpacing, 393
 TickMarkSpacing, 393
 Top, 97, 393
 TopLeftCell, 179
 Type, 143, 147, 396
 UsedRange, 240
 Users, 244
 Value, 213
 VerticalAlignment, 388
 Visible, 171, 213, 240
 VisibleFields, 327
 VisibleItems, 352
 Weight, 389
 Width, 392
 Worksheets, 223
 XML, 433
 ZOrder, 179, 379
 ZOrderPosition, 436

Семейство, 111

Axes, 384
 Characters, 441
 CommandBars, 145
 Comments, 240
 CustomViews, 228
 DataLabels, 408
 Dialogs, 162
 Names, 213
 OLEObjects, 177
 PivotFields, 319

Семейство (*продолжение*)

- PivotItemList, 361
- Points, 410
- RecentFiles, 201
- SeriesCollection, 400, 406
- Sheet, 112
- Sheets, 223, 229
- SmartTagRecognizers, 432
- Styles, 229
- UserAccessList, 245
- UserAccessObjects, 244
- UserForm, 113
- Windows, 214
- Workbooks, 216

Сервер автоматизации, 467

Смарт-теги, 431

Событие, 136

- AppEvents, 137
- BeforeClose, 37, 41
- BeforePrint, 37
- ChartEvents, 137
- cmdActivate_Click, 175
- DblClick, 176

Событие (*продолжение*)

- DocEvents, 137
- Initialize, 176, 247
- KeyDown, 176
- NewSheet, 37
- OLEObjectEvents, 137
- Open, 37, 123
- RefreshEvents, 137
- SelectionChange, 137
- Workbook_Open, 128, 132
- WorkbookEvents, 137

События, 37

Т

Типы

- диаграмм, 379

Точечная диаграмма, 382

Точка останова, 49

У

Узел, 35

Условное форматирование, 292

Уточненное имя процедуры, 88

Ф

Форма, 167

«Функция
UBound», 73

«Функция», 33

Функция

- AutoRecover, 482
- CInt, 84
- CountCharacters, 86
- CreateObject, 469
- GetObject, 469
- Immediate If, 96
- InchesToPoints, 97
- InputBox, 92
- InStr, 93
- InStrRev, 93
- IsDate, 95
- IsEmpty, 96
- IsMissing, 84
- IsNull, 96
- IsNumeric, 96
- LCase, 93
- Left, 93
- Len, 92
- LTrim, 94
- Mid, 93
- MsgBox, 90
- Now, 198
- PointsToInches, 98
- Quick Watch, 56
- Replace, 93
- Right, 93
- RTrim, 94
- Space, 94
- Str, 94
- StrCmp, 95
- String, 94
- Switch, 96
- TimeValue, 197
- Trim, 94
- UBound, 201
- UCase, 93
- Val, 94

Х

Хост-приложение, 475

Ц

Цилиндрическая диаграмма, 381

Ч

Члены объекта, 111

Ш

Шаблоны, 126

Шаг

с выходом, 55

с заходом, 55

с обходом, 55

Э

Элемент управления

ActiveX, 171

CommandBarControl, 147

FileOpen, 153

Элемент управления (*продолжение*)

msoControlButton, 144

msoControlPopup, 144

Spelling, 153

программный

идентификатор, 180

Элементы управления, 143

Я

Язык программирования

BASIC, 473

C, 475

C++, 476

COBOL, 478

FORTRAN, 478

Pascal, 477

Visual Basic, 474

Visual C++, 477

низкого уровня, 471

С. Роман

Использование макросов в Excel

2-е издание

Перевела с английского И. Рузмайкина

Главный редактор	<i>Е. Строганова</i>
Заведующий редакцией	<i>И. Корнеев</i>
Руководитель проекта	<i>А. Чижова</i>
Литературный редактор	<i>А. Сергеев</i>
Художник	<i>Н. Биржаков</i>
Корректоры	<i>Н. Викторова, И. Смирнова</i>
Верстка	<i>А. Келле-Пелле</i>

Лицензия ИД № 05784 от 07.09.01.

Подписано в печать 20.11.03. Формат 70×100/16. Усл. п. л. 41,28.

Тираж 4000 экз. Заказ № 1094.

ООО «Питер Принт». 196105, Санкт-Петербург, ул. Благодатная, д. 67в.

Налоговая льгота – общероссийский классификатор продукции

ОК 005-93, том 2; 953005 – литература учебная.

Отпечатано с готовых диапозитивов в ФГУП «Печатный двор» им. А. М. Горького
Министерства РФ по делам печати, телерадиовещания и средств массовых коммуникаций.

197110, Санкт-Петербург, Чкаловский пр., 15.

КЛУБ ПРОФЕССИОНАЛ

В 1997 году по инициативе генерального директора Издательского дома «Питер» Валерия Степанова и при поддержке деловых кругов города в Санкт-Петербурге был основан «Книжный клуб Профессионал». Он собрал под флагом клуба профессионалов своего дела, которых объединяет постоянная тяга к знаниям и любовь к книгам. Членами клуба являются лучшие студенты и известные практики из разных сфер деятельности, которые хотят стать или уже стали профессионалами в той или иной области.

Как и все развивающиеся проекты, с течением времени книжный клуб вырос в «Клуб Профессионал». Идею клуба сегодня формируют три основные «клубные» функции:

- неформальное общение и совместный досуг интересных людей;
- участие в подготовке специалистов высокого класса (семинары, пакеты книг по специальной литературе);
- формирование и высказывание мнений современного профессионала (при встречах и на страницах журнала).

КАК ВСТУПИТЬ В КЛУБ?

Для вступления в «Клуб Профессионал» вам необходимо:

- ознакомиться с правилами вступления в «Клуб Профессионал» на страницах журнала или на сайте www.piter.com;
- выразить свое желание вступить в «Клуб Профессионал» по электронной почте postbook@piter.com или по тел. (812) 103-73-74;
- заказать книги на сумму не менее 500 рублей в течение любого времени или приобрести комплект «Библиотека профессионала».

«БИБЛИОТЕКА ПРОФЕССИОНАЛА»

Мы предлагаем вам получить все необходимые знания, подписавшись на «Библиотеку профессионала». Она для тех, кто экономит не только время, но и деньги. Покупая комплект — книжную полку «Библиотека профессионала», вы получаете:

- скидку 15% от розничной цены издания, без учета почтовых расходов;
- при покупке двух или более комплектов — дополнительную скидку 3%;
- членство в «Клубе Профессионал»;
- подарок — журнал «Клуб Профессионал».

Закажите бесплатный журнал «Клуб Профессионал».

ИЗДАТЕЛЬСКИЙ ДОМ
ПИТЕР[®]
WWW.PITER.COM

КНИГА-ПОЧТОЙ



**ЗАКАЗАТЬ КНИГИ ИЗДАТЕЛЬСКОГО ДОМА «ПИТЕР»
МОЖНО ЛЮБЫМ УДОБНЫМ ДЛЯ ВАС СПОСОБОМ:**

- по телефону: (812) 103-73-74;
- по электронному адресу: postbook@piter.com;
- на нашем сервере: www.piter.com;
- по почте: 197198, Санкт-Петербург, а/я 619
ЗАО «Питер Пост».

**ВЫ МОЖЕТЕ ВЫБРАТЬ ОДИН ИЗ ДВУХ СПОСОБОВ ДОСТАВКИ
И ОПЛАТЫ ИЗДАНИЙ:**

-  Наложенным платежом с оплатой заказа при получении посылки на ближайшем почтовом отделении. Цены на издания приведены ориентировочно и включают в себя стоимость пересылки по почте (**но без учета авиатарифа**). Книги будут высланы нашей службой «Книга-почтой» в течение двух недель после получения заказа или выхода книги из печати.
-  Оплата наличными при курьерской доставке (**для жителей Москвы и Санкт-Петербурга**). Курьер доставит заказ по указанному адресу в удобное для вас время в течение трех дней.

ПРИ ОФОРМЛЕНИИ ЗАКАЗА УКАЖИТЕ:

- фамилию, имя, отчество, телефон, факс, e-mail;
- почтовый индекс, регион, район, населенный пункт, улицу, дом, корпус, квартиру;
- название книги, автора, код, количество заказываемых экземпляров.

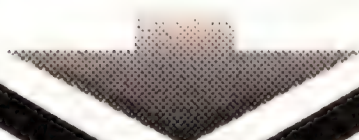
Вы можете заказать бесплатный журнал «Клуб Профессионал».

ИЗДАТЕЛЬСКИЙ ДОМ
 **ПИТЕР®**
WWW.PITER.COM

Нет времени ходить по магазинам?



наберите:



www.piter.com



Здесь вы найдете:

Все книги издательства сразу
Новые книги — в момент выхода из типографии
Информацию о книге — отзывы, рецензии, отрывки
Старые книги — в библиотеке и на CD



**И, наконец, вы нигде не купите
наши книги дешевле!**

ПРЕДСТАВИТЕЛЬСТВА ИЗДАТЕЛЬСКОГО ДОМА «ПИТЕР»
предлагают эксклюзивный ассортимент компьютерной, медицинской,
психологической, экономической и популярной литературы

РОССИЯ

Москва м. «Калужская», ул. Бутлерова, д. 17б, офис 207, 240; тел./факс (095) 777-54-67;
e-mail: sales@piter.msk.ru

Санкт-Петербург м. «Выборгская», Б. Сампсониевский пр., д. 29а;
тел. (812) 103-73-73, факс (812) 103-73-83; e-mail: sales@piter.com

Воронеж ул. 25 января, д. 4; тел. (0732) 27-18-86;
e-mail: piter-vrn@vmail.ru; piter@comch.ru

Екатеринбург ул. 8 Марта, д. 267б; тел./факс (3432) 25-39-94; e-mail: piter-ural@r66.ru

Нижний Новгород ул. Премудрова, д. 31а; тел. (8312) 58-50-15, 58-50-25;
e-mail: piter@infonet.nnov.ru

Новосибирск ул. Немировича-Данченко, д. 104, офис 502;
тел/факс (3832) 54-13-09, (3832) 47-92-93; e-mail: piter-sib@risp.ru

Ростов-на-Дону ул. Калитвинская, д. 17в; тел. (8632) 95-36-31, (8632) 95-36-32;
e-mail: jupiter@rost.ru

Самара ул. Новосадовая, д. 4; тел. (8462) 37-06-07; e-mail: piter-volga@sama.ru

УКРАИНА

Харьков ул. Суздальские ряды, д. 12, офис 10–11, т. (057) 712-27-05;
e-mail: piter@tender.kharkov.ua

Киев пр. Красных Казаков, д. 6, корп. 1; тел./факс (044) 490-35-68, 490-35-69;
e-mail: office@piter-press.kiev.ua

БЕЛАРУСЬ

Минск ул. Бобруйская д., 21, офис 3; тел./факс (37517) 226-19-53; e-mail: piter@mail.by

МОЛДОВА

Кишинев «Ауратип-Питер»; ул. Митрополит Варлаам, 65, офис 345; тел. (3732) 22-69-52,
факс (3732) 27-24-82; e-mail: lili@auratip.mldnet.com



Ищем зарубежных партнеров или посредников, имеющих выход на зарубежный рынок.
Телефон для связи: **(812) 103-73-73.**
E-mail: grigorjan@piter.com



Издательский дом «Питер» приглашает к сотрудничеству авторов.
Обращайтесь по телефонам: **Санкт-Петербург — (812) 327-13-11,**
Москва — (095) 777-54-67.



Заказ книг для вузов и библиотек: (812) 103-73-73.
Специальное предложение — e-mail: kozin@piter.com

УВАЖАЕМЫЕ ГОСПОДА!
КНИГИ ИЗДАТЕЛЬСКОГО ДОМА «ПИТЕР»
ВЫ МОЖЕТЕ ПРИОБРЕСТИ
ОПТОМ И В РОЗНИЦУ
У НАШИХ РЕГИОНАЛЬНЫХ ПАРТНЕРОВ.

Башкортостан

Уфа, «Азия», ул. Зенцова, д. 70 (оптовая продажа),
маг. «Оазис», ул. Чернышевского, д. 88,
тел./факс (3472) 50-39-00.
E-mail: asiaufa@ufanet.ru

Дальний Восток

Владивосток, «Приморский торговый дом книги»,
тел./факс (4232) 23-82-12.
E-mail: bookbase@mail.primorye.ru

Хабаровск, «Мирс»,
тел. (4212) 30-54-47, факс 22-73-30.
E-mail: sale_book@bookmirs.khv.ru

Хабаровск, «Книжный мир»,
тел. (4212) 32-85-51, факс 32-82-50.
E-mail: postmaster@worldbooks.kht.ru

Европейские регионы России

Архангельск, «Дом книги»,
тел. (8182) 65-41-34, факс 65-41-34.
E-mail: book@atnet.ru

Калининград, «Вестер»,
тел./факс (0112) 21-56-28, 21-62-07.
E-mail: nshibkova@vester.ru
<http://www.vester.ru>

Северный Кавказ

Ессентуки, «Россы», ул. Октябрьская, 424,
тел./факс (87934) 6-93-09.
E-mail: rossy@kmw.ru

Сибирь

Иркутск, «ПродаЛитъ»,
тел. (3952) 59-13-70, факс 51-30-70.
E-mail: prodalit@irk.ru
<http://www.prodalit.irk.ru>

Иркутск, «Антей-книга»,
тел./факс (3952) 33-42-47.
E-mail: antey@irk.ru

Красноярск, «Книжный мир»,
тел./факс (3912) 27-39-71.
E-mail: book-world@public.krasnet.ru

Нижневартовск, «Дом книги»,
тел. (3466) 23-27-14, факс 23-59-50.
E-mail: book@nvartovsk.wsnet.ru

Новосибирск, «Топ-книга»,
тел. (3832) 36-10-26, факс 36-10-27.
E-mail: office@top-kniga.ru
<http://www.top-kniga.ru>

Тюмень, «Друг»,
тел./факс (3452) 21-34-82.
E-mail: drug@tyumen.ru

Тюмень, «Фолиант»,
тел. (3452) 27-36-06, факс 27-36-11.
E-mail: foliant@tyumen.ru

Челябинск, ТД «Эврика», ул. Барбюса, д. 61,
тел./факс (3512) 52-49-23.
E-mail: evrika@chel.surnet.ru

Татарстан

Казань, «Таис»,
тел. (8432) 72-34-55, факс 72-27-82.
E-mail: tais@bancorp.ru

Урал

Екатеринбург, магазин № 14,
ул. Челюскинцев, д. 23,
тел./факс (3432) 53-24-90.
E-mail: gvardia@mail.ur.ru

Екатеринбург, «Валео-книга»,
ул. Ключевская, д. 5,
тел./факс (3432) 42-56-00.
E-mail: valeo@etel.ru

Использование макросов в Excel



«Использование макросов в Excel» представляет собой введение в программирование на Excel VBA и содержит множество примеров решения задач, для выполнения которых недостаточно возможностей, обеспечиваемых пользовательским интерфейсом.

Книга написана лаконичным стилем, но тем не менее в ней вы найдете всю информацию, касающуюся Excel VBA, которая жизненно необходима в процессе написания макросов.

Материал рассчитан на пользователей, а также на программистов, незнакомых с объектной моделью Excel. Подробно в книге обсуждаются следующие вопросы:

- Visual Basic Editor и среда программирования Excel VBA — полнофункциональная интегрированная среда разработки программ Excel, обеспечивающая написание, выполнение, тестирование и отладку VBA-макросов;
- язык программирования VBA, являющийся единым для всех приложений, входящих в состав офисных пакетов Microsoft Office;
- объектная модель Excel, включающая новые объекты и вновь появившиеся элементы существующих объектов.

После внимательного прочтения этой книги вы сможете постичь азы программирования на языке Excel VBA, что, безусловно, повысит эффективность и производительность работы, в какой бы версии Microsoft Excel вы ни работали.



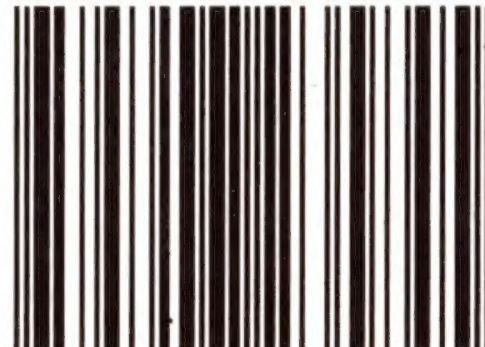
Категория: электронные таблицы

Уровень пользователя: опытный

Посетите web-сайт издательства O'Reilly: www.oreilly.com

Посетите наш web-магазин: <http://www.piter.com>

- ISBN 5-94723-584-6



9 785947 235845



O'REILLY

С. РОМАН

ИСПОЛЗОВАНИЕ МАКРОСОВ В EXCEL

2-Е ИЗДАНИЕ

